VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

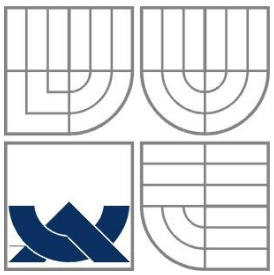
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EMULÁTOR KONCOVÝCH PRVKŮ INTELIGENTNÍ DOMÁCNOSTI

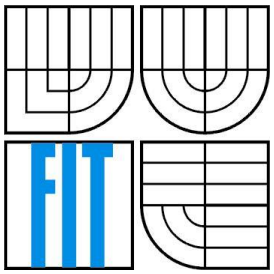
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE FILIP ŠUTOVSKÝ
AUTHOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EMULÁTOR KONCOVÝCH PRVKŮ INTELIGENTNÍ DOMÁCNOSTI

EMULATOR FOR INTELLIGENT HOME END STATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

FILIP ŠUTOVSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

ING. PAVOL KORČEK

BRNO 2015

Abstrakt

Tato práce se zabývá návrhem a implementací aplikace emulující koncové prvky systému inteligentní domácnosti, které slouží k měření veličin a interakci se spínači. Aplikace disponuje komplexním grafickým uživatelským rozhraním vyvíjeným v jazyce Java umožňující testování ostatních vrstev systému a zaznamenávání statistických dat o emulaci přispívajících k analýze systému.

Abstract

This thesis deals with design and implementation of an application emulating end stations in intelligent home system, which are used to measure physical quantities and interact with switches. Application has a comprehensive graphical user interface developed in programming language Java, which allows testing of other system layers and recording of statistical data about emulation contributing to system analysis.

Klíčová slova

grafické uživatelské rozhraní, emulátor, inteligentní domácnost, síťová komunikace, testování, JavaFX

Keywords

graphical user interface, emulator, intelligent home, network communication, testing, JavaFX

Citace

Šutovský Filip: Emulátor koncových prvků inteligentní domácnosti, bakalářská práce, Brno, FIT VUT v Brně, 2015

Emulátor koncových prvků inteligentní domácnosti

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Pavola Korčeka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Filip Šutovský
19.05.2015

Poděkování

Chtěl bych poděkovat vedoucímu mé práce Ing. Pavlovi Korčekovi za odborné vedení, seznámení s vyvíjeným systémem a pomoc při vypracovávaní bakalářské práce. Dále bych chtěl poděkovat mému konzultantovi Ing. Viktorovi Pušovi, Ph.D a celé výzkumné skupině, která se podílí na vývoji systému inteligentní domácnosti.

Obsah

1 Úvod

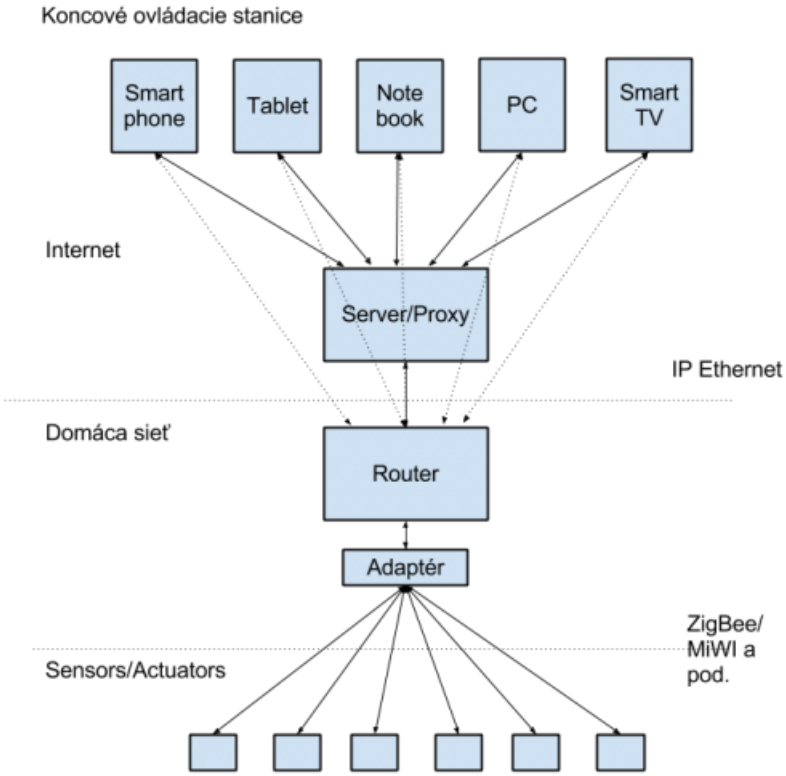
V dnešnej dobe sme svedkami rýchleho nástupu nového trendu, ktorým je takzvaný internet vecí. Ten predstavuje sieť, v ktorej sú navzájom prepojení nielen ľudia a počítače, ale aj procesy, dáta a objekty okolo nás. Jednotlivé zariadenia komunikujú s inými zariadeniami, prostrediami a infraštruktúrami. Výsledkom je obrovské množstvo dát spracovaných do užitočných akcií, ktoré nám môžu poskytnúť jednoduchší, bezpečnejší život a taktiež napríklad zmenšiť náš negatívny vplyv na životné prostredie. [1] K internetu budú pripojené a s jeho pomocou ovládané nielen počítače, mobilné telefóny či televízie, ale napríklad aj práčky, chladničky, automobily, či celé domy. Niekdajšia predstava sveta, kde sa začne variť voda na kávu v momente, kedy ráno zazvoní budík, a kde sa klimatizácia z ekologických dôvodov sama vypne pri opustení miestnosti, je dnes realitou. Takéto riešenia zabezpečujú inteligentné domy a domácnosti. Inteligentné domy ponúkajú vysokú úroveň automatizácie a ovládania, no ich zásadným problémom je nutnosť plánovať infraštruktúru systému, rozmiestnenie kabeláže a jednotlivých senzorov, už pri samotnej stavbe domu. Naopak inteligentné domácnosti je možné zriadiť v akomkoľvek prostredí, vďaka bezdrôtovej komunikácií a jednoduchej inštalácii.

Keďže vytvorenie systému inteligentnej domácnosti je komplexnou úlohou, vedie od návrhu a implementácie nízkoúrovňového hardvéru pre jednotlivé senzory a zariadenia spracovávajúce dáta zo senzorov, až k implementácií užívateľských ovládacích staníc, napr. aplikácie pre inteligentný telefón. Pri takomto procese nie je možné naraz zaručiť dostupnosť všetkých prvkov systému, hlavne hardvérových, a preto je vhodné niektoré z nich softvérovo emulovať. Úlohou tejto práce je navrhnúť a implementovať takýto emulátor, ktorý ponúkne vývojárom možnosť testovať ostané časti systému bez nutnosti existencie reálnych prvkov zaznamenávajúcich stav domácnosti. Ďalej bude zabezpečovať podrobné informácie a štatistiky o emulácii, vďaka ktorým bude možné dohľadať nedostatky v návrhu systému, či chyby v jeho implementácii. Aplikácia by mala poskytovať užívateľovi prehľadné rozhranie s možnosťou podrobného nastavenia, škálovania emulovaných prvkov a zároveň pohodlné používanie, ktoré prispieva k efektívnosti testovania.

V kapitole 2 je predstavený koncept a architektúru systému vyvíjaného na FIT VUT a všeobecne popisujem činnosti jeho jednotlivých častí. Ďalej podrobne rozoberám princíp a zabezpečenie komunikácie, aktuálne používané komunikačné protokoly a typy emulovaných prvkov. Kapitola 3 začína uvedením požiadavkou na výslednú aplikáciu. Nasleduje proces návrhu jednotlivých častí aplikácie s podrobným popisom ich funkcií a použitých prostriedkov. Zároveň v tejto kapitole uvádzam príklady použitia výslednej aplikácie. V nasledujúcej kapitole 4 popisujem samotnú implementáciu, v ktorej sa predovšetkým zameriavam na technické prevedenie hlavných aplikačných detailov a nástrojov potrebných na emuláciu. V závere predstavím použitie emulátora v praxi. V záverečnej kapitole 5 sumarizujem moje riešenie a navrhujem možné vylepšenia výslednej aplikácie.

2 Systém vyvíjaný na FIT VUT

Experimentálny systém inteligentnej domácnosti pozostáva z niekoľkých vrstiev (viď Obrázok 2.1 - Architektúra systému). Spodná vrstva je tvorená dvoma typmi zariadení. Senzory dodávajú vyšším vrstvám architektúry informácie o domácnosti a aktuátory ponúkajú možnosť interakcie. Ďalšiu vrstvu tvorí adaptér napojený na smerovač v domácnosti a slúži ako komunikačná brána medzi spodnou vrstvou architektúry a serverom. Server zabezpečuje medzivrstvu v komunikácii domácnosti s koncovými ovládacími stanicami. K jeho úlohám patrí ukladanie prijatých dát do databáze, poskytovanie prístupu k týmto dátam koncovým ovládacím staniciam, no v neposlednom rade aj spracovávanie nakumulovaných dát užívateľom zvolenými algoritmami. Vyhodnotenie týchto algoritmov môže viesť k prednastaveným akciám, ako napríklad upozornenie užívateľa či zmena stavu aktuátoru. Najvyššou vrstvou architektúry sú koncové ovládacie stanice, ktoré dávajú užívateľovi možnosť zobrazovať namerané hodnoty, či ovládať jednotlivé prvky domácnosti. [2]



Obrázok 2.1 - Architektúra systému.

2.1 Senzory a aktuátory

Senzory sú určené na snímanie veličín, ako napríklad teplota, tlak, vlhkosť a iné. Na druhej strane aktuátory zasahujú do prostredia, napríklad zapnutie spínača. Oba prvky komunikujú s adaptérom bezdrôtovo pomocou protokolu *MiWi* od spoločnosti *Microchip*, pričom ich dosah musí vyhovovať komunikácii v bežnom dome, či byte, musí byť odolný voči rušeniu ostatných spotrebičov a v neposlednom rade musí byť komunikácia šifrovaná kvôli možným narušiteľom. Každý prvok sa jedinečnou identifikáciou jednoznačne prihlási, či registruje v procese spárovania. Tento proces sa môže opakovať a musí prebehnúť bez zložitého zásahu užívateľa, napríklad stlačením tlačidla. Koncový prvok v ňom poskytne základné informácie o sebe pre vyššie vrstvy na neskoršie

zabezpečenie správnej komunikácie. Senzory budú mať možnosť sa po úspešnom spárovaní pravidelne uspať na zvolený časový interval a následne vykonať činnosť. Interval uspania môže byť zmenený vyššou vrstvou architektúry. Na druhú stranu aktuátory sa nikdy neuspávajú, aby boli schopné prijímať správy o zmene ich stavu. Väčšina koncových prvkov je napájaných pomocou batérie, ktorej proces uspávania zabezpečí dlhšiu výdrž. Samotné prvky budú o aktuálnej kapacite batérie informovať užívateľa, aby bol schopný včas vykonať výmenu. Taktiež môžu v systéme existovať aj prvky, ktoré budú napájané priamo zo zdroja elektrickej energie a môžu slúžiť na opakovanie bezdrôtového signálu, tzv. retranslačné stanice.

2.2 Adaptér

Adaptér je externé zariadenie napojené na smerovač v domácnosti a slúži ako brána pre komunikáciu koncových prvkov cez internet. Každý adaptér má jedinečnú identifikáciu, ktorou sa registruje do vyššej vrstvy, teda na server, a následne s ňou komunikuje. Ďalej zabezpečuje spárovanie s koncovými prvkami systému, dokáže ich jednoznačne adresovať a pomocou ich komunikačného protokolu s nimi komunikovať. Vďaka prevodu informácií z koncových prvkov cez adaptér je možné pridávať do systému prvky tretích strán, ktoré využívajú vlastný komunikačný protokol. Ak nastane výpadok pripojenia k internetu, adaptér je schopný zabezpečiť čiastočnú logiku operácií nad systémom, aby nedošlo k potenciálne nechceným, či nebezpečným situáciám a zároveň lokálne ukladá informácie o stave systému, ktoré pri obnovení spojenia odošle na server.

2.3 Server

Server slúži ako medzivrstva medzi adaptérom a koncovými ovládacími stanicami. Obsahuje logiku aplikácie inteligentnej domácnosti, databázové úložisko dát a logovacích súborov, vďaka ktorým sa môžu poskytovať služby využívajúce ich analýzu. Výhodou tohto riešenia je jednoduchosť ovládania koncových zariadení prihlásením do distribuovanej služby, kde si užívateľ zaregistruje adaptér, ktorý následne komunikuje priamo so serverom. Jeden užívateľ môže mať zaregistrovaných niekoľko adaptérov, pričom dokáže ovládať všetky z jednej koncovej ovládacej stanice.

2.4 Koncové ovládacie stanice

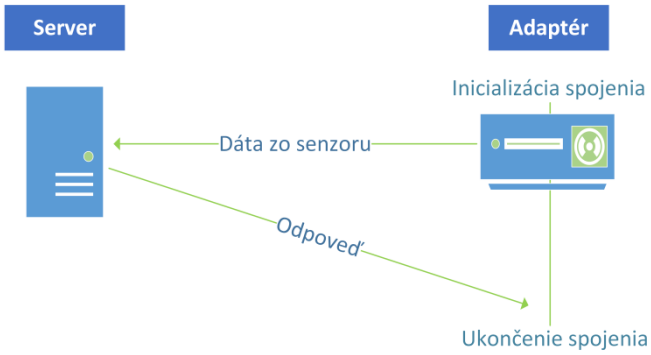
Koncové zariadenia umožňujú užívateľovi zobraziť hodnoty namerané senzormi, či prepínať aktuálne stavy aktuátorov. Ďalej umožňujú meniť intervaly zasielania informácií u jednotlivých koncových prvkov, prepínať medzi adaptérmi, ak užívateľ disponuje viacerými, a spravovať užívateľské kontá či ich prístup k jednotlivým adaptérom. Taktiež ponúkajú rozhranie na definovanie algoritmov, ktoré spracovávajú dáta z domácnosti a nastaviť vykonanie akcie po ich vyhodnotení. Ako koncové zariadenie môže byť použité klasické PC, inteligentný telefón, tablet, či SmartTV.

2.5 Komunikácia

Pre komunikáciu s vyššou vrstvou architektúry, teda serverom, sa využíva výmena správ vo formáte *XML*. Je to vhodný spôsobov výmeny dát medzi zariadeniami kvôli jeho univerzálnym vlastnostiam ako napríklad platformová nezávislosť, dobrá prenositeľnosť, formát čitateľný človekom i strojom, natívna podpora jazykovej lokalizácie, hierarchická štruktúra vhodná pre väčšinu druhov dát a ďalšie.

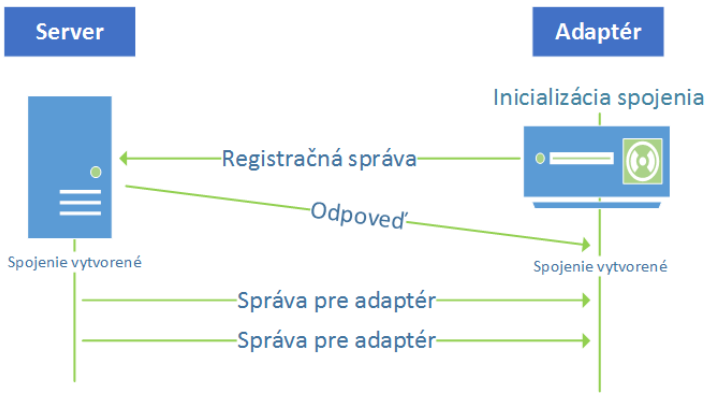
2.5.1 Priebeh komunikácie

Server komunikuje s adaptérom pomocou dvoch samostatných spojení. Prvé spojenie je synchrónne a dočasné, pričom je vyvolané prebudením senzora zo spánkového intervalu. Začína inicializovaním spojenia zo strany adaptéra, zaslaním správy o danom senzore, prijatím odpovede servera a následným ukončením spojenia (viď Obrázok 2.2). Adaptér po prijatí správy kontroluje zmenu intervalu uspania a v prípade nekonzistencie s aktuálnym tento interval aktualizuje. Nakoniec sa senzor opätovne uspí.



Obrázok 2.2 - Grafické zobrazenie dočasného spojenia.

Druhé spojenie je stále a vytvára sa pri spustení adaptéra. Slúži na zasielanie asynchrónnych správ zo strany servera, resp. koncových ovládacích staníc. Podobne ako pri predchádzajúcom spojení začína inicializáciou spojenia zo strany adaptéra, ktorý následne zašle registračnú správu. Po prijatí odpovede od servera o úspechu registrácie začína adaptér prijímať správy na príslušnom porte (viď Obrázok 2.3). Tento proces sa opakuje po každom výpadku spôsobenom chybou spojenia či reštarte adaptéra.



Obrázok 2.3 - Grafické zobrazenie stáleho spojenia.

2.5.2 Zabezpečenie komunikácie

Výmenu správ v komunikácii je nutné zabezpečiť pred krádežou, prečítaním či úpravou informácií treťou stranou. V dnešnej dobe sú najbežnejšie používanými štandardmi SSL (Secure Sockets Layer) a jeho nástupca TLS (Transport Layer Security). Oba kryptografické protokoly poskytujú možnosť zabezpečenej komunikácie cez internet, pričom sú medzi nimi len drobné rozdiely.

Pomocou kryptografie poskytujú svojim koncovým bodom autentizáciu na základe certifikátov. Všeobecne môžu systémy využívať jednostrannú autentizáciu, kedy je zaručená identita servera, zatiaľ čo klient zostáva neautorizovaný. Druhou možnosťou je obojstranná autentizácia, pri ktorej obe strany majú istotu, s kým komunikujú. [3]

V momentálnej dobe je v komunikácii medzi serverom a adaptérom inteligentnej domácnosti využívaný protokol TLS s jednostrannou autentizáciou a certifikátom typu X.509.

2.5.3 Komunikačné protokoly

Na server odosiela adaptér hodnoty získané zo senzoru spolu s informáciami nutnými k identifikácii daného adaptéra ako napríklad sériové číslo adaptéra posielajúceho správu, verzia firmvéru adaptéra poskytujúca vzdialenú aktualizáciu, verzia komunikačného protokolu medzi adaptérom a serverom zaisťujúca možnosť dodatočnej úpravy komunikácie. Ďalej obsahuje časovú pečiatku na identifikáciu postupnosti jednotlivých správ a informácie o jednotlivých senzoroch (viď Príklad 1 - Formát správy zasielanej na server.).

Každý senzor poskytuje svoje jedinečné identifikačné číslo, hodnotu nabitia batérie, silu signálu a hodnoty samotných meraných veličín popísaných typom veličiny a odsadením v rámci tohto senzoru. Odsadenie sa používa pre identifikáciu veličín rovnakého typu v jednom senzore (viď Príklad 1 - Formát správy zasielanej na server.). Táto správa je zasielaná využitím dočasného spojenia so serverom (viď Kapitola 2.5.1).

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter_server protocol_version="0.1" state="data"
adapter_id="64996" fw_version="0" time="1403173390">
    <device id="01234567">
        <debug protocol_version="0.1" fw_version="1"/>
        <battery value="40"/>
        <rssi value="80"/>
        <values>
            <value type="0x00" offset="0x00">25.5</value>
            <value type="0x00" offset="0x01">26.0</value>
            <value type="0x01" offset="0x00">67.0</value>
        </values>
    </device>
</adapter_server>
```

Príklad 1 - Formát správy zasielanej na server.

Server odpovedá správou, ktorá obsahuje verziu komunikačného protokolu a jedinečnú identifikáciu senzoru v rámci domácnosti. Ďalej obsahuje, v prípade senzoru, čas ďalšieho zobudenia a odoslania novej správy na server v sekundách (viď Príklad 2 – Odpoveď servera na správu s dátami.).

```
<?xml version="1.0" encoding="UTF-8"?>
<server_adapter protocol_version="0.1" state="update" id="01234567"
time="5"/>
```

Príklad 2 – Odpoveď servera na správu s dátami.

Ďalšou časťou protokolu je registračná správa, ktorú využíva adaptér po pripojení stáleho spojenia (viď Kapitola 2.5.1). Obsahuje verziu komunikačného protokolu medzi adaptérom a serverom, jedinečné identifikačné číslo adaptéra spolu s jeho verziou firmvéru a časovú pečiatku pokusu o registráciu (viď Príklad 3 – Registračná správa.).

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter_server protocol_version="0.1" state="register"
adapter_id="64996" fw_version="0.0" time="1403173390"/>
```

Príklad 3 – Registračná správa.

Server odpovedá správou o úspechu prípadne neúspechu registrácie. Táto správa obsahuje verziu komunikačného protokolu a samotnú odpoveď (viď Príklad 4 - Odpoveď servera na registráciu.).

```
<?xml version="1.0" encoding="UTF-8"?>
<server_adapter protocol_version="0.1" state="register"
response="true"/>
```

Príklad 4 - Odpoveď servera na registráciu.

Zvyšné správy sú inicializované zo strany servera a využívajú stále spojenie (viď Kapitola 2.5.1). Na žiadnu z týchto správ adaptér neodpovedá. Patrí medzi ne správa informujúca adaptér o prepnutí do stavu párovania, v ktorom dokáže pripájať nové senzory do svojej siete. Správa obsahuje verziu komunikačného protokolu a jedinečný identifikátor adaptéra (viď Príklad 5 - Správa o párovaní.).

```
<?xml version="1.0" encoding="UTF-8"?>
<server_adapter protocol_version="0.1" state="listen" id="64996"
time="0"/>
```

Príklad 5 - Správa o párovaní.

Nasledujúcou správou je informovanie aktuátoru o zmene stavu. Obsahuje verziu komunikačného protokolu, jedinečné identifikačné číslo zariadenia v rámci domácnosti a nové hodnoty aktuátorov označené typom a odsadením v rámci tohto zariadenia (viď Príklad 6 - Správa o zmene stavu aktuátora.). Adaptér túto správu spracuje a zašle signál na zmenu stavu na zariadenie.

```
<?xml version="1.0" encoding="UTF-8"?>
<server_adapter protocol_version="0.1" state="set" id="01234567"
time="0">
    <value type="0xA0">1</value>
    <value type="0xA5">27.0</value>
</server_adapter>
```

Príklad 6 - Správa o zmene stavu aktuátora.

Poslednou správou protokolu je požiadavka na odstránenie zariadenia zo siete adaptéra. Obsahuje verziu komunikačného protokolu adaptéra so serverom a jedinečné identifikačné číslo zariadenia, ktoré má byť odstránené (viď Príklad 7 - Správa o odstránení senzora.).

```
<?xml version="1.0" encoding="UTF-8"?>
<server_adapter protocol_version="0.1" state="clean" id="01234567"
time="0"/>
```

Príklad 7 - Správa o odstránení senzora.

Na identifikáciu typov meraných veličín či aktuátorov v správach, resp. v celom systéme inteligentnej domácnosti, sa využíva tabuľka typov. Definuje označenie veličiny, názov veličiny, jednotku veličiny a dátový typ používaný na ich manipuláciu v rámci aplikácií. Pri aktuátoroch navyše definuje veľkosti ich hodnôt (viď Tabuľka 1 a Tabuľka 2).

Typy emulovaných senzorov a ich označenie			
Typ	Význam	Jednotky	Použitý typ
0x0A	Teplota	°C	*Float*
0x01	Vlhkosť	%	*Integer*
0x02	Tlak	hPa	*Integer*
0x03	Zopnutie - senzor	Open/close	*Boolean*
0x04	Zopnutie - prepínač	ON/OFF	*Boolean*
0x05	Intenzita svetla	1x	*Float*
0x06	Intenzita hluku	dB	*Float*
0x07	Emisie (CO_2)	ppm	*Integer*
0x08	Pozícia		*Bit array*
0x0A	Teplota	°C	*Float*
0x0B	Stav kotla		*Integer*

Tabuľka 1 - Typy emulovaných veličín a ich označenie.

Typy emulovaných aktuátorov a ich označenie				
Typ	Význam	Jednotky	Použitý typ	Veľkosť
0xA0	Zopnutie prepínač (on/off)		*Boolean*	*1b*
0xA1	Zopnutie (on-only)		*Boolean*	*1b*
0xA2	Prepnutie (toggle)		*Boolean*	*1b*
0xA3	Rozsah		*Integer*	*1B*
0xA4	Farba (RGB)		*Integer*	*3B*
0xA5	Teplota	°C	*Float*	*4B*
0xA6	Typ operácie kotla		*Integer*	*1B*
0xA7	Režim operácie kotla		*Integer*	*1B*

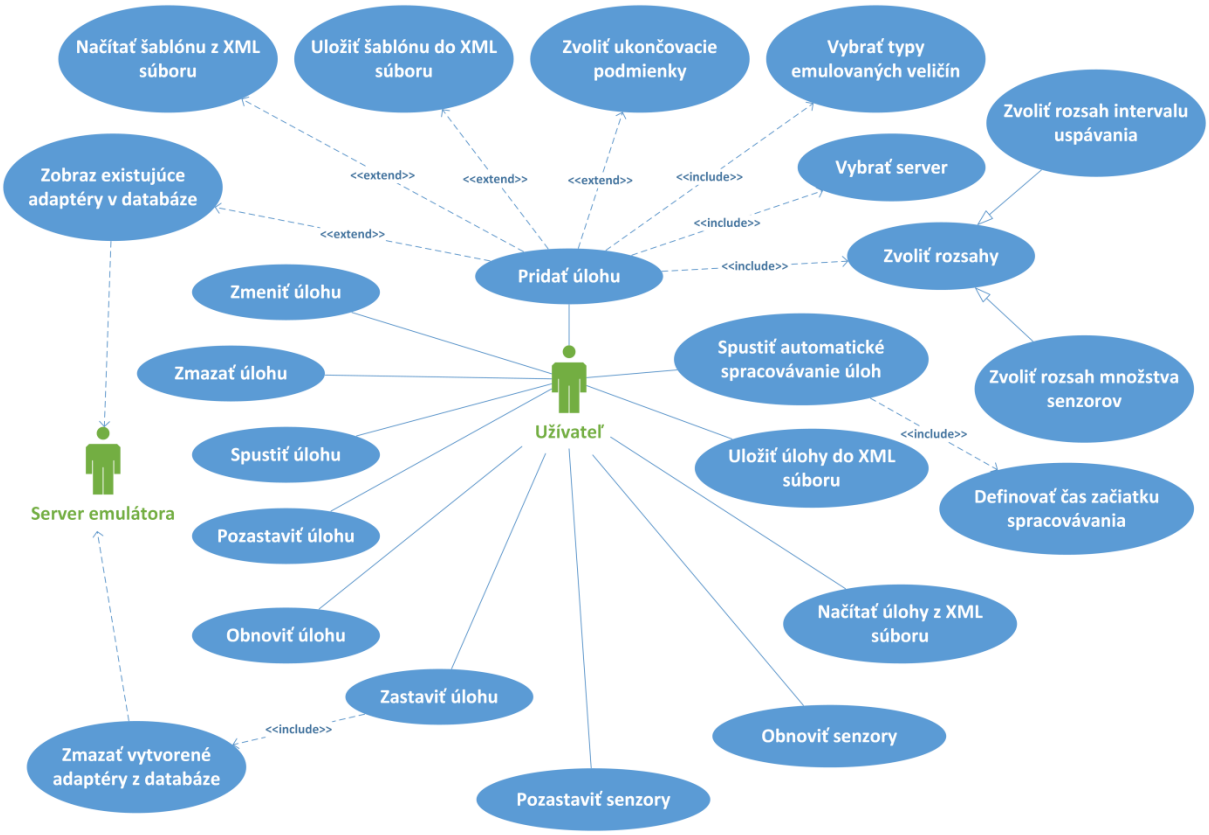
Tabuľka 2 - Typy emulovaných aktuátorov a ich označenie.

3 Vlastná aplikácia

Výsledná aplikácia emulátora by mala byť schopná podrobne otestovať koncové ovládacie stanice a zároveň hraničné možnosti zaťaženia serverovej časti architektúry. Preto by mala spĺňať niekoľko požiadaviek, medzi ktoré patrí využívanie komunikačných protokolov a princípov komunikácie, ktoré sa zhodujú s reálnymi prvkami. Ďalej by mali byť, z pohľadu vyšších vrstiev architektúry, emulované prvky nerozoznateľné od reálnych, a preto je potreba podrobne napodobiť ich správanie. V neposlednom rade musí byť možné meniť a ukladať nastavenia emulovaných prvkov v užívateľskom rozhraní.

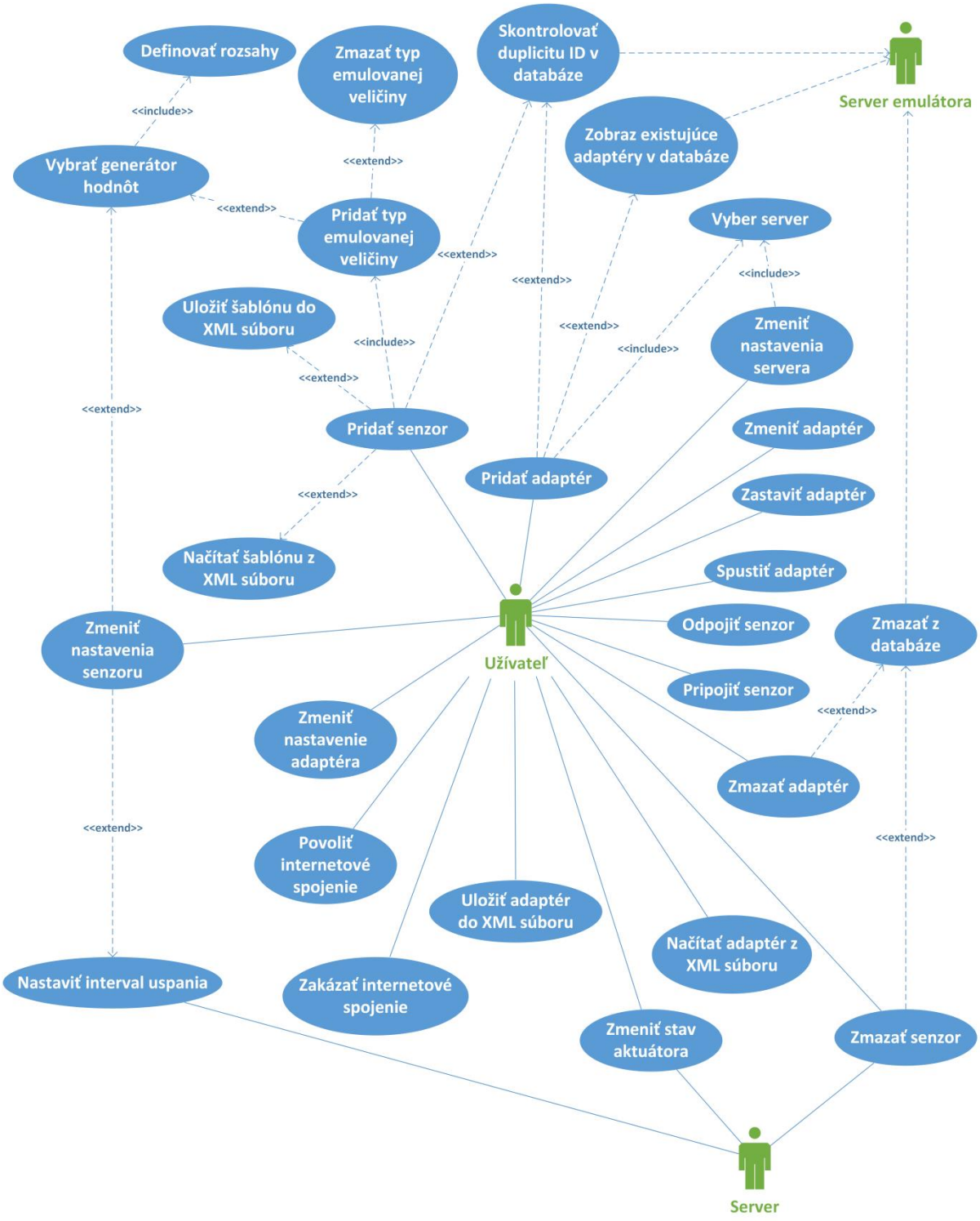
3.1 Návrh

Pre splnenie požiadaviek na emulátor musí byť aplikácia schopná otestovať rôzne vrstvy architektúry. Každá s týchto vrstiev vyžaduje samostatný prístup k zobrazovaniu a ovládaniu emulovaných dát, a preto bude aplikácia rozdelená na dve časti. Prvou časťou aplikácie je výkonnostná simulácia poskytujúca užívateľovi možnosť pridania či mazania úloh, ktoré vytvárajú veľké množstvo emulovaných prvkov. Týmto aplikácia umožňuje vytvorenie veľkého množstva požiadaviek na server, a tým pádom je možné zaznamenať jeho chovanie v hraničných situáciách. Ďalšie akcie výkonnostnej simulácie, zobrazené v diagrame užitia (viď Obrázok 3.1), ponúkajú možnosť ovládania, bližšej špecifikácie, no i ukladania vytvorených úloh.



Obrázok 3.1 - Diagram prípadov užitia výkonnostnej simulácie.

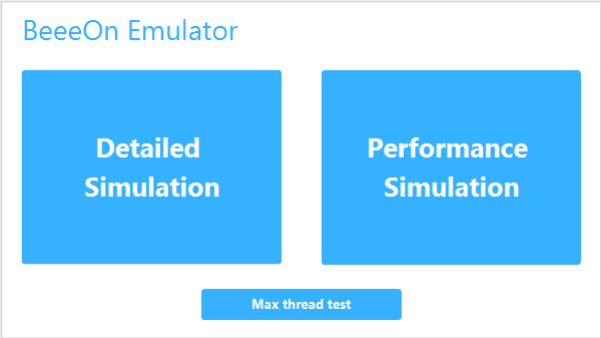
Druhou časťou je detailná simulácia, ktorá poskytne užívateľovi podrobný náhľad na emulované prvky a ich ovládanie. Jednotlivé akcie, zobrazené v diagrame prípadov užitia (viď Obrázok 3.2), slúžia na vytváranie, mazanie a hlavne podrobné nastavovanie prvkov a veličín, ktoré emulujú. Taktiež možno na diagrame vidieť aké akcie môžu byť vykonané zo strany servera, respektíve koncových ovládacích staníc.



Obrázok 3.2 - Diagram prípadov užitia detailnej simulácie.

3.2 Užívateľské prostredie

Pri spustení aplikácie má užívateľ na výber z dvoch druhov simulácie (viď Obrázok 3.3). Detailná a výkonnostná simulácia. Každá sa zameriava na testovanie inej vyššej vrstvy architektúry systému, no obe využívajú rovnaké princípy emulovania zariadení.



Obrázok 3.3 - Úvodný dialóg aplikácie.

Každý emulovaný adaptér má za úlohu správu senzorov/aktuátorov, uchovávanie správ senzorov čakajúcich na odoslanie, pridávanie informácií o adaptéri do správ pre server podľa protokolov, vykresľovanie zmien stavov adaptéra, informovanie užívateľa o priebehu komunikácie, doposiaľ neodoslaných správach či chybových hláseniach, pomocou zapisovania do logovacej sekcie aplikácie. Na rozdiel od stávajúceho riešenia adaptér neslúži iba na prevod správ medzi senzormi a serverom. Prijaté správy od servera adaptér rozbalí a príslušným senzorom/aktuátorom mení jednotlivé atribúty, čím simuluje zasielanie správ adaptérom na dané senzory. Navyše ponúka možnosť výpisu a ukladania času odozvy servera na jednotlivé požiadavky, či počet odoslaných správ, vďaka čomu zabezpečuje dôkladnejšie informácie o testovaní pre užívateľa. Ďalej oznamuje o nesprávnych či porušených správach od servera a následné vypnutie senzoru, ktorému bola správa zaslaná, aby sa predišlo opakovanému chybovému stavu a nekonzistencii atribútov senzoru.

Emulovaný senzor ponúka možnosť pripojenia k adaptéru a neskoršieho odpojenia od adaptéra, ukladanie histórie hodnôt emulovaných veličín, metódy na úpravu informácií o senzore, na vytvorenie správ pre server podľa protokolu a rozhranie na zmenu stavov aktuátorov. Jednou z informácií o senzore/aktuátore je hodnota a typ emulovanej veličiny. Táto hodnota môže byť zadaná užívateľom alebo generovaná aplikáciou. Pre všetky typy veličín sa môže nastaviť niekoľko druhov generovania novej hodnoty, pričom užívateľ pri vytváraní senzora poskytne potrebné informácie (napr. maximálna hodnota, minimálna hodnota, atď.).

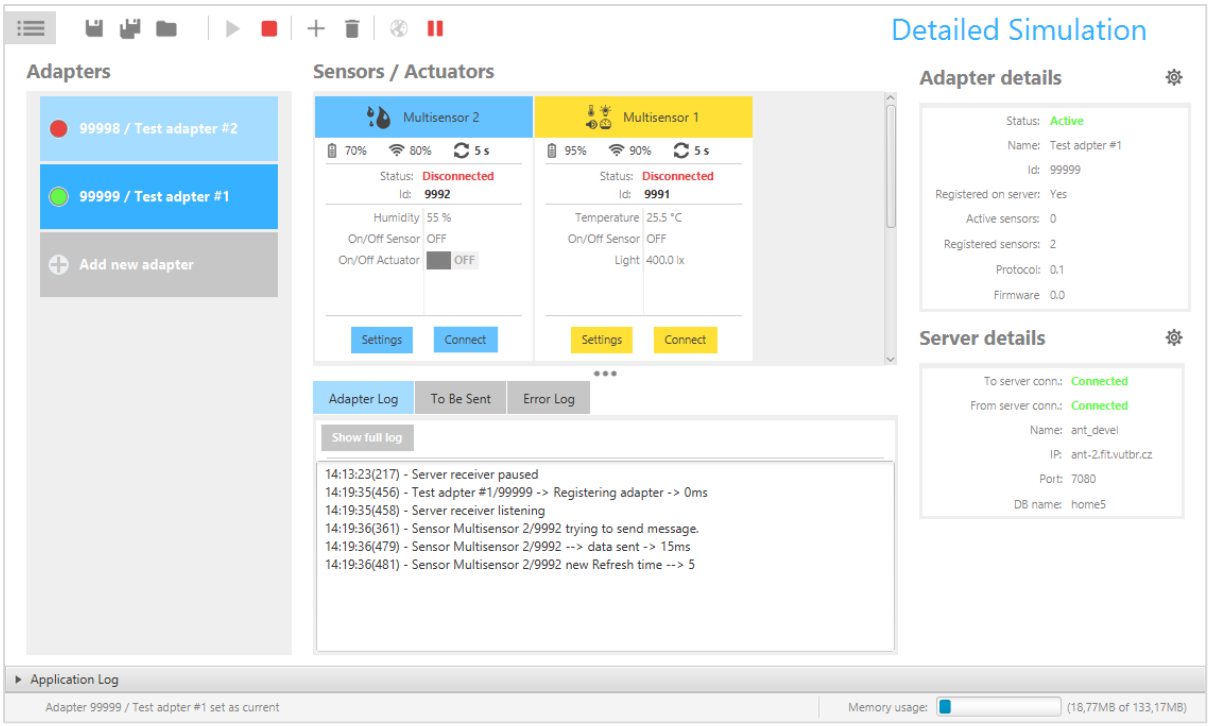
Adaptéry komunikujú so serverom dočasným spojením, kedy sa po vytvorení spojenia odošle správa, následne sa prijme odpoveď a spojenie sa ukončí. Tento druh komunikácie je vyhovujúci pre senzory merajúce veličiny, pretože užívateľ nemôže meniť ich hodnotu. Na druhej strane podľa architektúry systému inteligentnej domácnosti sa aktuátory nikdy neuspávajú, aby boli vždy prístupné zmene stavu pomocou koncovej ovládacej stanice, a preto je nutné vytvoriť ďalšie spojenie. Toto spojenie je inicializované každým adaptérom, pričom sa nikdy neukončuje (vynímajúc chybový stav spojenia) a prijíma požiadavky na zmenu stavov aktuátorov z vyšších vrstiev architektúry (viď Kapitola 2.5.1).

Aplikácia taktiež poskytuje možnosť simulovania výpadku internetové spojenia, kedy sú správy odosielané senzormi ukladané na adaptér a pri obnovení spojenia prednostne odoslané na server. Užívateľ dokáže vďaka tejto funkcii testovať schopnosť servera udržiavať konzistenciu

informácií o adaptéroch a ich senzoroch/aktuátoroch. Aby sa predišlo nekonzistencii dát v databáze pri zadávaní duplicitných identifikátorov adaptérov či senzorov, je pridaná funkcia kontroly existencie identifikátora v databáze.

3.3 Detailná simulácia

Detailná simulácia je zameraná na testovanie koncových ovládacích staníc. Vďaka detailnému popisu a možnosti meniť nastavenia každého vytvoreného adaptéra a všetkých jeho senzorov a aktuátorov jednotlivo, ponúka užívateľovi možnosť vytvárať konkrétne testovacie situácie.



Obrázok 3.4 - Návrh detailnej simulácie.

V ľavej časti aplikácie sa nachádza panel obsahujúci tlačidlá všetkých doposiaľ vytvorených adaptérov označených jedinečným identifikačným číslom, názvom zadaným pri vytváraní a indikátorom aktivity. Zároveň obsahuje tlačidlo na pridanie nového adaptéra. Navigácia prebieha kliknutím na spomínané tlačidlá, kedy aplikácia na zvyšok okna vykreslí všetky informácie vzťahujúce sa k danému adaptéru a jeho senzorom, rozdelené do štyroch hlavných oblastí.

Prvá oblasť sa nachádza v pravom dolnom rohu okna aplikácie (viď Obrázok 3.4 - Návrh detailnej simulácie.). Obsahuje informácie o serveri, s ktorým adaptér komunikuje a indikátory spojenia so serverom. Prvý indikátor sa vzťahuje k dočasnému spojeniu používanému na pravidelné zasielanie správ zo senzorov. Je farebne odlíšený od zvyšku informácií a jeho stav je určený úspechom alebo neúspechom doručenia poslednej odoslanej správy. Druhý indikátor sa vzťahuje k stálemu spojeniu používanému na prijímanie správ pre aktuátory z vyšších vrstiev architektúry. Podobne ako prvý indikátor, je farebne odlíšený od zvyšku informácií.

Druhá oblasť sa nachádza v spodnej strednej časti okna aplikácie (viď Obrázok 3.4 - Návrh detailnej simulácie.) a ponúka tri druhy logovacích výpisov:

- komunikačné výpisy - podrobný popis akcií vykonávaných senzormi/aktuátormi ako napríklad pokus o odoslanie správy, oznámenie o uspaní senzora, zmena intervalu uspania a

ďalšie. Taktiež uvedomuje užívateľa o úspešnom predaní správy a prijatí odpovede, spolu s časom potrebným na túto komunikáciu,

- výpisy čakajúcich správ - zaznamenávanie údajov o doposiaľ neodoslaných správach,
- chybové výpisy - zaznamenávanie chybových stavov adaptéra a jeho senzorov/aktuátorov.

Tretia oblasť sa nachádza v pravej časti okna aplikácie (viď Obrázok 3.4 - Návrh detailnej simulácie.). Jej súčasťou je, podobne ako pri detailoch servera v prvej oblasti, zreteľne vyznačený indikátor aktivity adaptéra, meniteľný priradenými tlačidlami v navigácii aplikácie. Navyše, pre obmedzené možnosti grafického prostredia zobraziť všetky senzory/aktuátory naraz, informuje o počte vytvorených a aktuálne aktívnych senzoroch/aktuátoroch, čím zabezpečuje prehľad užívateľa o existujúcich adaptéroch.

Posledná, štvrtá, oblasť sa nachádza v hornej strednej časti okna aplikácie (viď Obrázok 3.4 - Návrh detailnej simulácie.). Obsahuje grafickú reprezentáciu všetkých senzorov/aktuátorov vytvorených vybraným adaptérom a tlačidlo na vytvorenie nového senzoru/aktuátoru. Okrem detailov, ako identifikačné číslo, kapacita batérie, kvalita signálu, indikátor aktivity meniteľný priradeným tlačidlom a interval uspania, zobrazuje aktuálnu hodnotu všetkých meraných veličín. Pre aktuátory zobrazuje tlačidlo na rýchlu zmenu stavu. Ak je zvolené ukladanie týchto hodnôt, užívateľ má možnosť zobraziť históriu v podobe histogramu.

Za účelom kvalitnejšieho testovania systému a prirodzenejšieho používania aplikácie využíva prostredie ukladanie a opätovné načítanie informácií o simulácii vo formáte XML, ktorý som zvolil pre jeho prehľadnosť a možnosť zmeniť obsah bez nutnosti použitia vyvíjanej aplikácie. [4] Je možné uložiť buď aktuálne vybraný adaptér do samostatného súboru, alebo všetky adaptéry vytvorené v dobe behu aplikácie kolektívne (viď Príloha B). Spolu s dátami potrebnými pre neskoršie načítanie sa taktiež uchovávajú semienka generátorov emulovaných veličín.

Generátor využíva semienko ako štartovaciu pozíciu pre svoj algoritmus pseudonáhodných čísiel, čím zabezpečuje produkovanie rovnakej sekvencie generovaných hodnôt a tým pádom umožňuje aplikácii presne replikovať testovaciu situáciu. [5]

Ďalšou prednosťou podporujúcou testovanie systému, po ukladaní simulácie, je automatické ukladanie logovacích výpisov (viď Príloha D). Po vytvorení nového adaptéra sa zároveň vytvorí súbor na disku, s dopredu nastaveným umiestnením, kde sa počas behu aplikácie zaznamenávajú výpisy daného adaptéra. Týmto spôsobom je zabezpečená spätná kontrola priebehu emulácie i po závažnej chybe, či zastavení aplikácie.

Obrázok 3.5 - Navigácia detailnej simulácie.

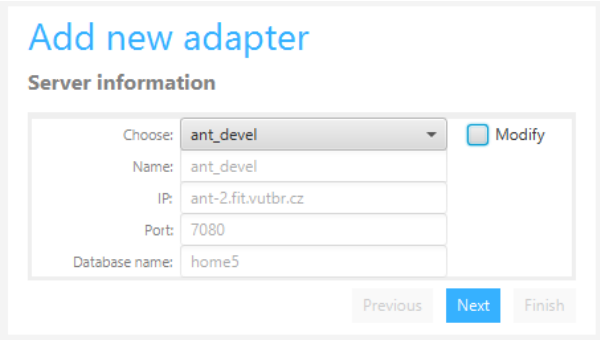
Aktuálne vybratý adaptér je riadený niekoľkými tlačidlami v navigácii aplikácie (viď Obrázok 3.5 - Navigácia detailnej simulácie.), taktiež prístupných pomocou klávesových skratiek, ktorých funkcionalita je nasledovná:

- rolovacie menu – poskytuje slovne popísané tlačidlá s rovnakou funkciou ako nasledovné tlačidlá,
- uloženie adaptéra - uloží aktuálne vybratý adaptér do samostatného *XML* súboru,
- uloženie adaptérov - uloží všetky vytvorené adaptéry do jedného *XML* súboru,
- načítanie adaptéra/adaptérov - načítanie adaptérov uložených v *XML* súbore,

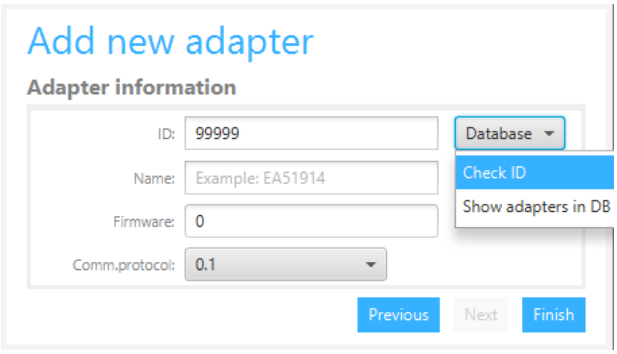
- aktivácia adaptéra - aktivuje aktuálne vybratý adaptér a senzory k nemu pripojené, čím spustí zasielanie správ na server,
- zastavenie adaptéra - deaktivuje aktuálne vybratý adaptér, čím zastaví odosielanie správ na server a odpojí stále spojenie,
- pridanie senzoru - zobrazí dialógové okno pre pridanie nového senzoru na aktuálne vybratý adaptér,
- zmazanie senzoru - zobrazí dialógové okno pre zmazanie senzorov aktuálne vybratého adaptéra z aplikácie. Taktiež ponúka možnosť automatického zmazanie z databáze,
- povolenie internetového spojenia - zastaví simuláciu výpadku internetového spojenia pre aktuálne vybratý adaptér,
- zakázanie internetového spojenia – povolí simuláciu výpadku internetového spojenia pre aktuálne vybratý adaptér.

3.3.1 Pridanie nového adaptéra

Pri pridávaní adaptéra sa zobrazí sprievodné modálne[1] dialógové okno, kde v prvej časti užívateľ nastaví informácie o serveri, pričom má možnosť výberu z prednastavených serverov alebo pridanie vlastného servera (viď Obrázok 3.6). V druhej časti sprievodcu užívateľ určí identifikačné číslo, názov, verziu firmvéru a verziu komunikačného protokolu (viď Obrázok 3.7). Pred pridaním má užívateľ možnosť overiť duplicitu zadaného identifikačného čísla v databáze, či zobraziť prehľad všetkých existujúcich identifikačných čísiel. Vďaka tejto funkcii je možné predísť nekonzistencii dát v databáze. Vyplnené informácie sú pred pridaním overované, čím sa užívateľ vyhne možnej chybe pri zadávaní.



Obrázok 3.6 - Pridanie nového adaptéra, výber serveru.

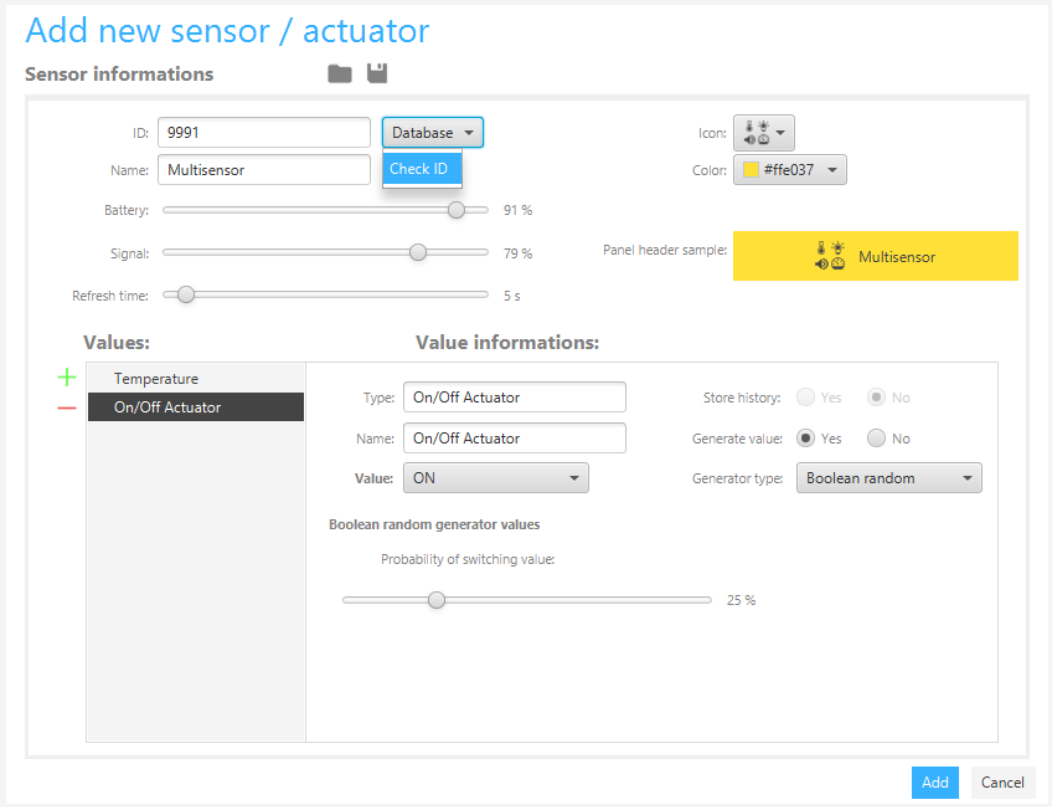


Obrázok 3.7 - Pridanie nového adaptéra, výber nastavení.

[1] Užívateľ je nútený uzavrieť modálne dialógové okno, aby sa mohol vrátiť k ovládaniu zvyšku aplikácie.

3.3.2 Pridanie nového senzoru

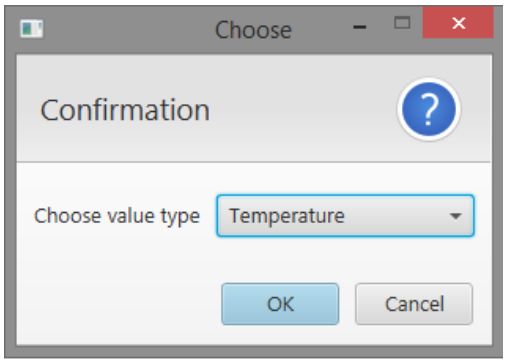
Pri pridávaní senzoru sa zobrazí modálne dialógové okno (viď Obrázok 3.8), kde užívateľ zadá všetky informácie potrebné na emulovanie daného senzoru. Jednou z hlavných údajov je jedinečné identifikačné číslo. Rovnako ako pri pridávaní adaptéra (viď Kapitola 3.3.1) má užívateľ možnosť kontroly duplicity v databáze, aby nevznikla nekonzistencia dát.



Obrázok 3.8 - Pridanie nového senzoru.

Ďalej užívateľ zvolí kapacitu batérie, hodnotu signálu a iniciálny interval uspania. Taktiež môže vybrať obrázok a farbu senzoru, ktoré sa zobrazujú v aplikácii po jeho vytvorení.

Nakoniec je potreba zvoliť emulované veličiny. Po stlačení príslušného tlačidla na pridanie novej hodnoty sa zobrazí ďalšie dialógové okno (viď Obrázok 3.9), v ktorom užívateľ vyberie jeden z implementovaných typov veličín (viď Kapitola 2.5.3).



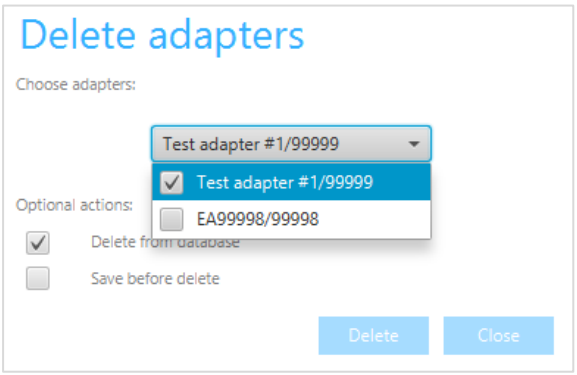
Obrázok 3.9 - Výber emulovanej veličiny.

Po potvrdení výberu sa v strednej oblasti prvého dialógového okna (viď Obrázok 3.8) zobrazia podrobné informácie o vybratej veličine, kde môže užívateľ zvoliť iniciálnu hodnotu, povolenie generovania nových hodnôt a samotný spôsob generovania. Tento proces sa opakuje pre každú zvolenú emulovanú veličinu.

Po vyplnení všetkých informácií má užívateľ možnosť uložiť šablónu s nastaveniami senzoru do formátu *XML*, ktorá sa môže neskôr slúžiť pre rýchle vytvorenie nového senzoru.

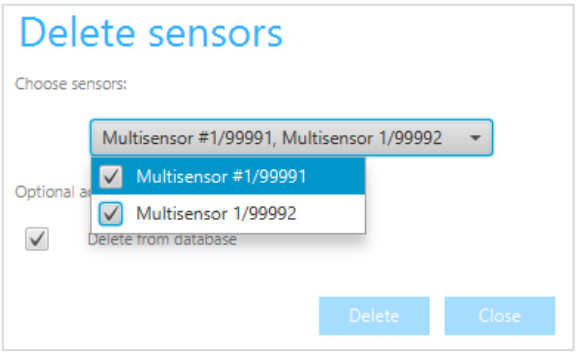
3.3.3 Odstránenie adaptéra a senzorov

Pri odstraňovaní adaptérov sa zobrazí dialógové okno (viď Obrázok 3.10) s výberom adaptérov, ktoré majú byť vymazané. Užívateľ môže zvoliť viacero adaptérov naraz. Navyše dialóg ponúka voliteľné možnosti. Vymazanie daných adaptérov z databáze a uloženia nastavení adaptéra do XML súboru pred vymazaním.



Obrázok 3.10 - Dialóg mazania adaptérov.

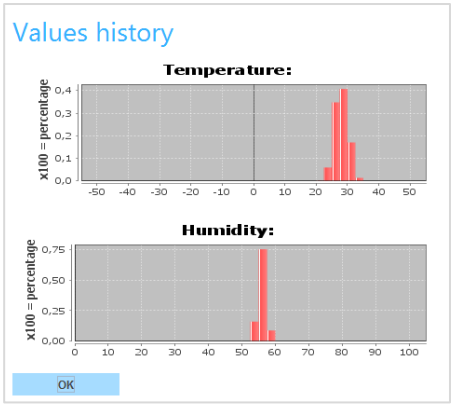
Dialóg pre odstránenie senzorov je obdobný ako dialóg odstránenia adaptérov. Ponúka možnosť vymazania viacero senzorov naraz a ich následné odstránenie z databáze (viď Obrázok 3.11).



Obrázok 3.11 - Dialóg mazania senzorov.

3.3.4 História hodnôt emulovaných veličín

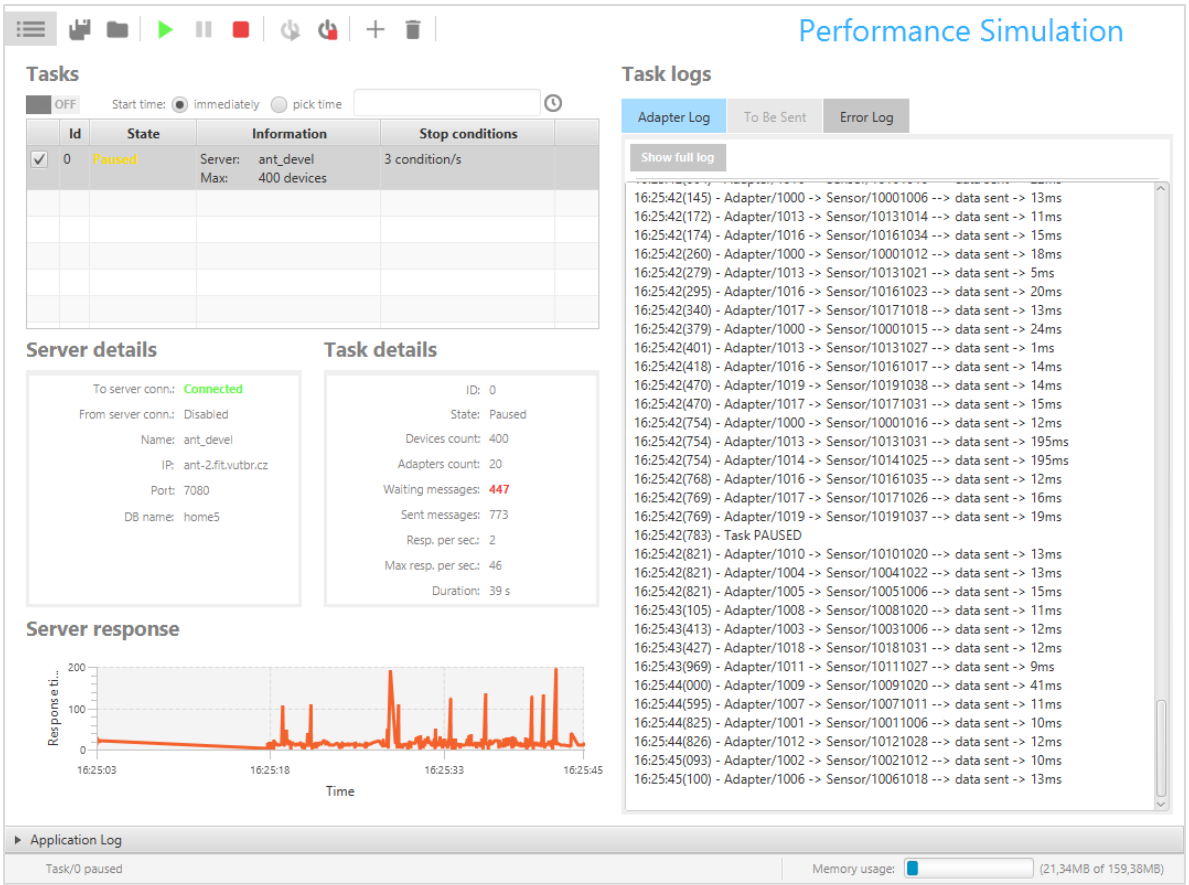
Zobrazenie histórie hodnôt prebieha v samostatnom dialógovom okne a slúži k porovnaniu dát získaných na koncových ovládacích staniciach architektúry systému inteligentnej domácnosti (viď Obrázok 2.1 - Architektúra systému). Všetkým emulovaným veličinám, s výnimkou typov udávajúcich pravdivosť, je generovaný histogram všetkých predchádzajúcich hodnôt (viď Obrázok 3.12 - História hodnôt emulovaných veličín.).



Obrázok 3.12 - História hodnôt emulovaných veličín.

3.4 Výkonnostná simulácia

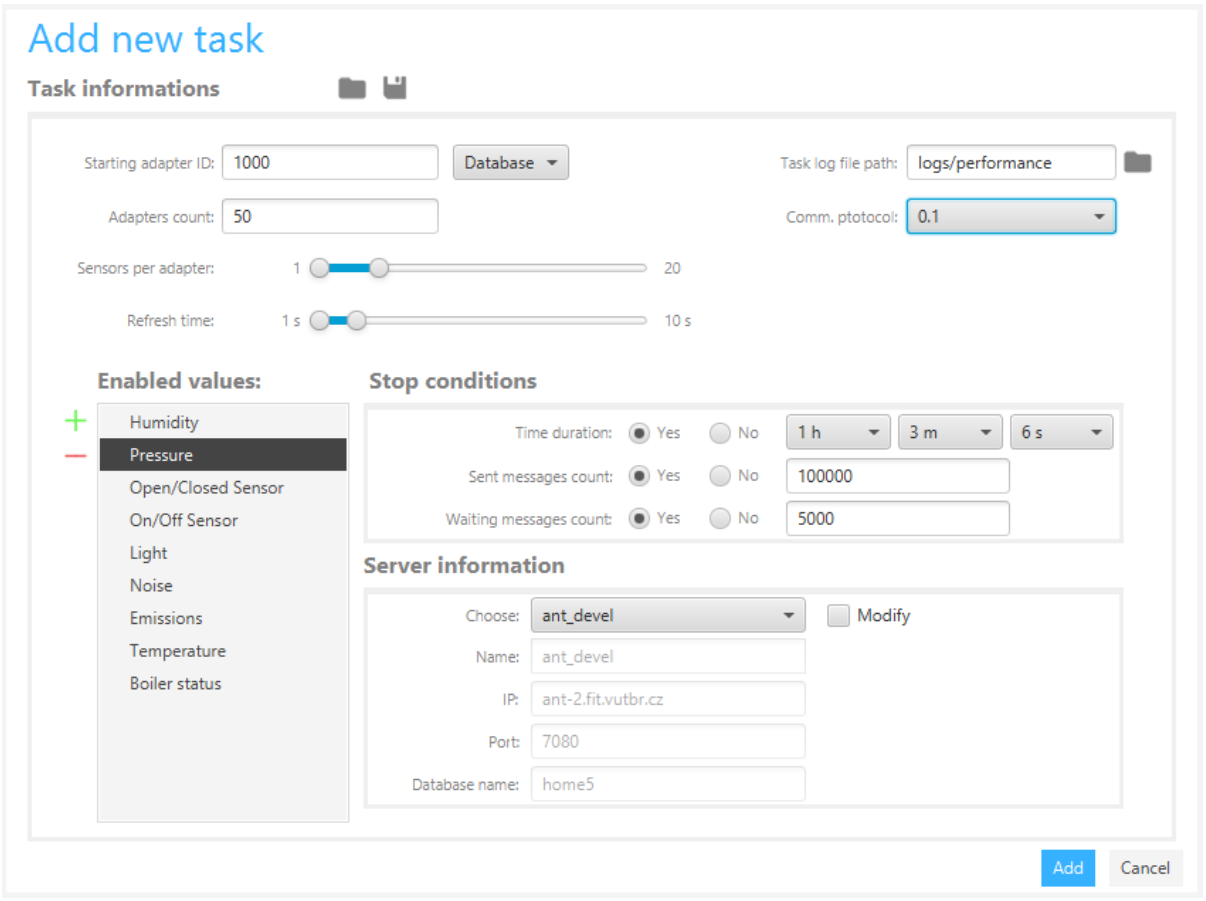
Výkonnostná simulácia je zameraná na testovanie serverovej časti architektúry (viď Obrázok 2.1 - Architektúra systému). Zameriava sa na generovanie veľkého množstva adaptérov a senzorov, pričom emulované veličiny sú generované náhodne, z dopredu zadaného rozsahu. Ďalej ponúka všeobecné štatistiky a grafy analyzované zo všetkých senzorov (viď Obrázok 3.13), ako napríklad priemerná doba odpovede serveru v danom čase, ukladanie aktuálnej simulácie a jej neskoršie načítanie, logovací súbor, či chybový súbor. Taktiež môže slúžiť ako doplnok k detailnej simulácii, kedy užívateľ skúma chovanie jednotlivých senzorov pri vyššom zaťažení servera.



Obrázok 3.13 - Výkonnostná simulácia.

V porovnaní s detailnou simuláciou (viď Kapitola 3.3), neexistuje možnosť náhľadu či ovládania adaptérov a senzorov jednotlivo. Pred spustením užívateľ vytvorí novú úlohu (viď Obrázok 3.14 - Pridanie novej úlohy.), kde určí všeobecné nastavenia, ktorými sa ďalej simulácia riadi. Tieto nastavenia nie je možné modifikovať počas behu simulácie z dôvodu náročnosti prípadných operácií a možného vzniku nekonzistencie dát uložených v databáze servera.

Medzi hlavné nastavenia patrí počet emulovaných adaptérov a rozsah množstva senzorov pre jednotlivé adaptéry. Jedno z ďalších nastavení je určenie spodnej a hornej hranice intervalu uspávania. Každému vytváranému senzoru sa priradí náhodne generovaná iniciálna doba uspania z tohto rozsahu, čím sa docieli rôzneho času odosielania správ serveru. Nasleduje definovanie detailov pripojenia k serveru a výber emulovaných veličín. Užívateľ má možnosť výberu typov emulovaných veličín, pričom aplikácia pri vytváraní senzoru náhodne zvolí jeden typ zo zadanej množiny a priradí ho tomuto senzoru. Každý vybratý typ, pri ktorom je to možné, musí mať definované rozsahy potrebné pre generátor hodnôt.



Obrázok 3.14 - Pridanie novej úlohy.

V neposlednom rade patrí medzi nastavenia novej úlohy určenie ukončovacích podmienok (viď Obrázok 3.15 - Nastavenie ukončovacích podmienok úlohy.), ktoré slúžia na automatické zastavenie úlohy. Užívateľ má na výber z troch možností:

- doba trvania úlohy - nastavenie času, po ktorom sa úloha ukončí,
- počet odoslaných správ - počet úspešne odoslaných správ na server, po dosiahnutí ktorého sa úloha ukončí,

- počet čakajúcich správ - počet správ čakajúcich na odoslanie na server, po dosiahnutí ktorého sa úloha ukončí. Ak je táto podmienka nastavená dostatočne veľká a je prekročená, môže to znamenať prílišné zaťaženie servera.

Obrázok 3.15 - Nastavenie ukončovacích podmienok úlohy.

Po vyplnení všetkých informácií o novej úlohe má užívateľ možnosť uložiť šablónu nastavení, ktorá sa môže neskôr použiť na rýchle vytvorenie úlohy s možnosťou modifikácie. Vytvorené úlohy sú zobrazené v tabuľke v ľavej hornej časti obrazovky (viď Obrázok 3.13 - Výkonnostná simulácia.). Užívateľ má možnosť ovládania úloh manuálne, pomocou tlačidiel umiestnených v navigácii aplikácie, no taktiež má možnosť spustiť úlohy automaticky v prednastavený čas. Táto funkcia má za úlohu odbremeniť užívateľa od manuálneho ovládania aplikácie pri zložitých a dlhotrvajúcich simuláciách. V tomto prípade však musí mať každá úloha minimálne jednu ukončovaciu podmienku, aby sa predišlo nekonečnému behu jednej úlohy.

Pod tabuľkou úloh sa nachádzajú panel s informáciami o serveri, na ktorý sa adaptéry vytvorené úlohou pripájajú, a farebne odlíšený indikátor spojenia. Taktiež sa v tejto oblasti nachádzajú detailné informácie o aktuálnej úlohe. Medzi tieto informácie patrí počet vytvorených adaptérov a senzorov, počítadlo odoslaných správ, počítadlo správ čakajúcich na odoslanie, priemerná odozva servera, maximálny počet správ odoslaných za jednu sekundu a čas trvania úlohy. Tieto informácie slúžia ako detailnejší pohľad užívateľa na bežiacu úlohu a zároveň ako štatistické údaje pre neskoršiu analýzu.

Ďalšou časťou aplikácie je graf zobrazujúci čas odozvy servera na jednotlivé správy senzorov. Vertikálna osa určuje dobu odozvy v milisekundách, horizontálna osa zobrazuje čas odoslania správy vo formáte hodina, minúta, sekunda. Tento graf pomáha napríklad pri identifikovaní správy, ktorá spôsobila pridlhú odpoveď servera.

Pravá oblasť okna aplikácie (viď Obrázok 3.13 - Výkonnostná simulácia.) poskytuje náhľad na dva druhy logovacích výpisov (viď Príloha E), rozdelených podobne ako pri detailnej simulácii:

- komunikačné výpisy - stručný popis akcií vykonávaných senzormi ako napríklad zmena intervalu uspania. Taktiež uvedomuje užívateľa o úspešnom predaní správy a prijatí odpovede, spolu s časom potrebným na túto komunikáciu,
- chybové výpisy - zaznamenávanie chybových stavov adaptérov, ich senzorov a simulácie ako celku.

Obrázok 3.16 - Navigácia výkonnostnej simulácie.

Aktuálna úloha je riadená niekoľkými tlačidlami v navigácii aplikácie (viď Obrázok 3.16 - Navigácia výkonnostnej simulácie.), taktiež prístupných pomocou klávesových skratiek, ktorých funkcionalita je nasledovná:

- rolovacie menu – poskytuje slovne popísané tlačidlá s rovnakou funkciou ako nasledovné tlačidlá,
- uloženie úlohy/úloh - uloženie vytvorených úloh do *XML* súboru,

- načítanie úlohy/úloh - načítanie úloh z *XML* súboru,
- štart úlohy/návrat k úlohe - z informácii zadaných o úlohe vytvorí požadovaný počet adaptérov a spustí ich komunikáciu so serverom. Ak je úloha pozastavená, aktivuje všetky adaptéry a začne znovu spracovávať správy pre server,
- pozastavenie úlohy - deaktivuje všetky vytvorené adaptéry, vďaka čomu pozastaví spracovávanie posielaných správ,
- zastavenie úlohy - na dopredu stanovené miesto uloží logovací súbor úlohy spolu so všetkými nazbieranými štatistickými údajmi o úlohe. Nasleduje odstránenie adaptérov z databáze a z emulátora, a nakoniec pripravenie prostredia na novú úlohu,
- obnovenie senzorov - aktivuje senzory všetkých vytvorených adaptérov,
- pozastavenie senzorov - deaktivuje senzory všetkých vytvorených adaptérov, čím zamedzí vytváranie nových správ a dáva možnosť odoslať správy čakajúce vo fronte,
- pridanie úlohy - zobrazí dialógové okno pre pridanie novej úlohy a
- zmazanie úlohy/úloh – zobrazí dialógové okne pre mazanie jednej či viacerých úloh.

Podobne ako pri detailnej simulácii prostredie automaticky ukladá logovacie sekcie úloh do súboru a ponúka uchovávanie, prípadne neskoršie načítanie úloh vo formáte *XML* (viď Príloha B). Keďže väčšina informácií o vytváraných adaptéroch je získavaná automaticky zo zadaných rozsahov, sú k nim priložené i semienka generátorov zabezpečujúce presnú replikáciu testovacej situácie.

3.5 Generovanie hodnôt emulovaných veličín

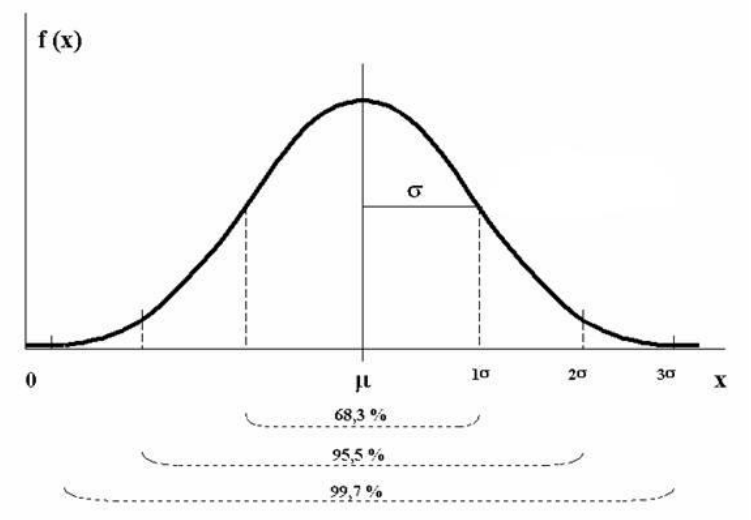
Pre modelovanie náhodných hodnôt numerických emulovaných veličín som sa rozhodol použiť dve možnosti, ktoré umožňujú sledovať zmeny na zariadeniach vo vyšších vrstvách architektúry. Prvou je normálne (Gaussovo) rozloženie, ktoré poskytuje hodnoty blížiace sa reálnym zmenám meraných veličín.

Gaussova krivka (hustota pravdepodobnosti) je funkcia závislá na dvoch premenných. Stredná hodnota μ a smerodajná odchýlka σ, ktoré charakterizujú variabilitu náhodnej veličiny X. [6]

$$X \sim N(\mu, \sigma^2) \qquad (1)$$

Grafickým znázornením je zvonovitá krivka (viď Obrázok 3.17), symetrická okolo strednej hodnoty μ ležiacej práve pod jej vrcholom. Tvar krivky napovedá, že pri opakovaní náhodného pokusu riadiaceho sa Gaussovým rozložením budú najčastejšie vychádzať výsledky v okolí strednej hodnoty. Rozptyl σ^2 určuje, ako tesne sa krivka primyká strednej hodnote. Podľa pravidla *troch sigma* leží výsledok náhodného pokusu s rozložením $N(\mu,\sigma^2)$ v intervale:

- 68,3% pravdepodobnosť rozmedzia hodnôt $\mu \pm 1\sigma$
- 95,5% pravdepodobnosť rozmedzia hodnôt $\mu \pm 2\sigma$
- 99,7% pravdepodobnosť rozmedzia hodnôt $\mu \pm 3\sigma$

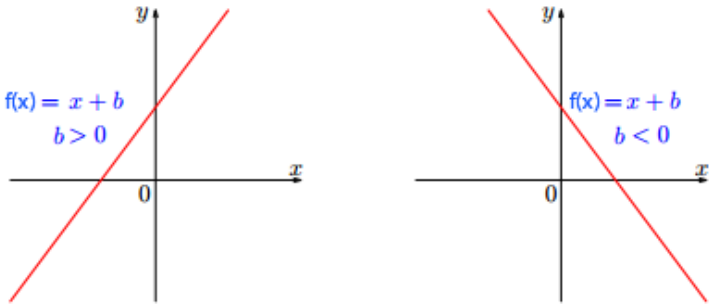


Obrázok 3.17 – Grafické znázornenie Gaussovho normálneho rozloženia pravdepodobnosti.

Druhou možnosťou generovania numerických hodnôt je lineárna funkcia, vďaka ktorej dokáže užívateľ rýchlo simulovať prechod veličiny do kritických hodnôt. Je závislá na dvoch premenných. Aktuálna hodnota *x* a konštanta *b*. Smernica funkcie *a* je vždy rovná 1.

$$f(x) = ax + b \quad (2)$$

Grafickým znázornením je priamka, ktorá je v závislosti na konštante *b* rastúca alebo klesajúca (viď Obrázok 3.18).



Obrázok 3.18 - Grafické znázornenie lineárnej funkcie.

3.6 Použité technológie

Aplikáciu som implementoval v programovacom jazyku *Java*, konkrétne verzia *Java SE8*, rozhranie pre programovanie aplikácií verzie *1.8.0_25*. Táto verzia ponúka mnoho vylepšení oproti predchádzajúcim. Medzi najvýznamnejšie s pohľadu mojej aplikácie patrí pokročilá správa vlákien a automatické synchronizovanie jednotlivých metód, ktoré zabraňuje vzniku nekonzistencie dát. Ďalšou príhodnou funkciou sú *lambda* výrazy, ktoré umožňujú používať funkcionalitu ako argument metódy alebo kód ako dáta. Týmto ponúkajú ľahšie čitateľný kód bez straty zložitosti výrazov. [7]

Keďže aplikácia nebude prístupná širokej verejnosti, ale úzkemu okruhu ľudí spolupracujúcich na vývoji inteligentnej domácnosti, skúsených v oblasti informačných technológií, nebolo nutné sa zameriavať na kompatibilitu s predchádzajúcimi verziami jazyku *Java* a mohol som naplno využívať možnosti, ktoré aktuálna verzia ponúka.

Grafické užívateľské prostredie aplikácie som tvoril pomocou platformy `JavaFX 2.2` obsiahnutej v balíčku `javafx` rozhrania pre programovanie aplikácií jazyku *Java*. Využil som základné triedy, ktoré ponúka, ako aj mnou implementované rozšírenia týchto tried, s účelom docieliť moderný a efektívny dizajn. Medzi výhody tejto platformy patrí deklaratívni značkovací jazyk FXML, založený na jazyku XML. Slúži k definovaniu komponentov a rozloženia užívateľského prostredia, pričom podporuje prispôsobovanie zobrazovania grafických komponentov pomocou jazyka CSS. Navyše `JavaFX` platforma ponúka nástroj `JavaFX Scene Builder` na rýchlu a jednoduchú tvorbu užívateľského rozhrania aplikácie bez nutnosti priameho písania zdrojového kódu. Umožňuje preťahovať rôzne komponenty do užívateľského rozhrania, jednoducho meniť ich vlastnosti a štýl. Výsledkom je FXML súbor kombinovateľný s kódom vývojára. [8]

Použité knižnice:

- controlsfx-8.20.8.jar, openjfx-dialogs-1.0.2.jar, jfxtras-common-8.0-r3.jar, jfxtras-control-8.0-r3.jar, jfxtras-fxml-8.0-r3.jar – voľne šíriteľné knižnice poskytujúce veľmi kvalitné grafické komponenty a iné nástroje k doplneniu platformy `JavaFX`,
- log4j-api-2.1.jar, log4j-core-2.1.jar - najrozšírenejšia voľne šíriteľná knižnica na vytváranie a jednoduchú správu logovacích správ. Umožňuje rozdelenie výpisov podľa aktuálnej potreby programátora,
- dom4j-2.0.0-ALPHA-2.jar – voľne šíriteľná knižnica pre prácu s XML, XPath a XSLT na *Java* platforme. Má plnú podporu DOM, SAX a JAXP definícií dokumentov.

4 Implementácia

V tejto kapitole popíšem jednotlivé časti implementácie mojej aplikácie, ktorú sú vyvíjal na operačnom systéme *Windows 8 64-bit* a linuxovej distribúcií *Fedora 20 „HeisenBug" 64-bit*.

4.1 Konfiguračný súbor aplikácie

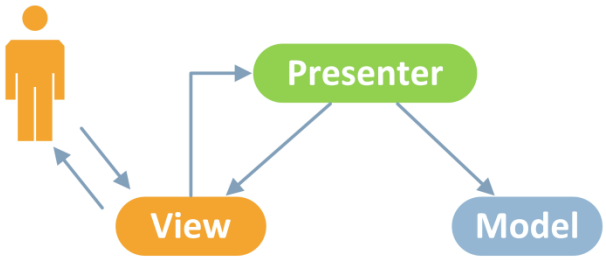
Aplikácia získava pri spustení niekoľko nastavení zo súboru s príponou `properties`. [9] Tento typ súboru sa využíva prevažne v jazyku Java a súvisiacich technológiách na uloženie konfigurovateľných parametrov. Každý parameter je uložený ako dvojica textových reťazcov oddelených špeciálnym znakom. Prvý reťazec, taktiež nazývaný kľúč, udáva meno parametru a druhý jeho hodnotu. Pomocou metód triedy `Properties`, obsiahnutej v základnom balíčku `java.util` programovacieho jazyka Java, aplikácia takýto súbor analyzuje a rozdelí do sady nastavení.

Nastavenia obsahujú verziu aplikácie, štandardnú verziu firmvéru emulovaných adaptérov, úroveň výpisov knižnice `log4j` (viď Kapitola 3.6) použitej na logovanie aplikácie a prednastavené informácie serverov, na ktoré sa emulované adaptéry môžu pripájať (viď Príloha A).

4.2 Grafické rozhranie

Pri vytváraní grafického rozhrania aplikácie som sa riadil návrhovým vzorom MVP (Model-View-Presenter), ktorý je deriváciou známeho návrhového vzoru MVC (Model-View-Controller) a je natívne podporovaný platformou JavaFX (viď Kapitola 3.6). MVP je rovnako ako MVC zameraný na separáciu konceptov prezentačnej logiky do troch častí (viď Obrázok 4.1), no jedným zo zásadných rozdielov je priama väzba View na Presenter:

- Model - je rozhranie definujúce dáta, ktoré majú byť zobrazené alebo inak spracované,
- View - je rozhranie na zobrazovanie, prípadne formátovanie, zobrazovaných dát a presmerovanie príkazov užívateľa na Presenter,
- Presenter – obsahuje aplikačnú a prezentačnú logiku. Manipuluje s Modelom, čo pomocou systému notifikácií zaistí aktualizáciu View , alebo ovlivňuje View priamo.



Obrázok 4.1 - Návrhový vzor MVP.

Vlastnosti tried, ktoré sú priamo zobrazované v užívateľskom rozhraní, sú definované pomocou rozhrania `Properties`, ktoré ponúka platforma JavaFX. Vďaka tomuto rozhraní môžu byť vlastnosti priamo naviazané na grafické komponenty, čím platforma zabezpečuje automatické vykresľovanie zmien, či možnosť načúvania týmto zmenám a vykonania prednastavenej akcie. [10]

Keďže sa aplikácia delí na dve samostatné časti (viď Kapitola 3.2), triedy grafického rozhrania sú rozdelené podobne, no navyše je pridané okno výberu časti aplikácie. Každé samostatné okno aplikácie je definované troma druhmi súborov.

Prvý súbor s názvom podľa vzoru `Názov.fxml` slúži na definovanie grafických komponentov a ich vlastností pomocou značkovacieho jazyka FXML platformy JavaFX (viď Kapitola 3.6). Druhý súbor s názvom podľa vzoru `NázovView.java` je naviazaný na prvý súbor a tým zabezpečuje prístup ku grafickým komponentom. Tretí súbor s názvom podľa vzoru `NázovPresenter.java` obsahuje logiku okna aplikácie, ktorá je pri jednotlivých oknách aplikácie nasledovná:

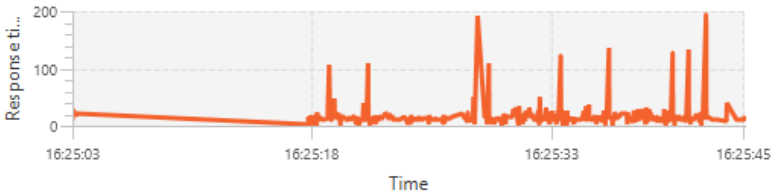
- `ChooserPresenter` – jednoduchý výber medzi detailnou a výkonnostnou simuláciou. Taktiež zabezpečuje načítavanie konfiguračného súboru aplikácie (viď Kapitola 4.1),
- `DetailedPresenter` – obsahuje logiku ovládania a zobrazovania detailnej simulácie, navyše udržuje zoznam emulovaných adaptérov,
- `PerformancePresenter` – obsahuje logiku ovládania a zobrazovanie výkonnostnej simulácie, navyše udržuje zoznam vytvorených úloh.

4.3 Jadro aplikácie

Jadro aplikácie tvorí niekoľko tried a rozhraní (viď Príloha C), ktoré sú využívané oboma časťami aplikácie (viď Kapitola 3). Pomocou ich vlastností a metód je možné emulovať činnosť a stavy adaptéra, senzorov, no navyše i kumulovať štatistické informácie o priebehu emulácie.

4.3.1 Zhromažďovanie informácií pre užívateľa

Trieda `AdapterLogger` slúži na zaznamenávanie a zobrazovanie výpisov ostatných tried do logovacej oblasti aplikácie (viď Obrázok 3.4 a Obrázok 3.13). Metódy výpisu sú synchronizované, aby sa predišlo možnému prepisovaniu informácií viacerými vláknami, a tým pádom nekonzistencii zaznamenávaných výpisov. Pri vytvorení inštancie tejto triedy sa zároveň vytvára nový súbor. Každý výpis sa zapisuje do tohto súboru a zároveň sa zobrazuje v užívateľskom prostredí, ktoré je však limitované pevne daným počtom výpisov, po ktorom sa logovacia oblasť aplikácie vyčistí. Vďaka tomuto systému zapisovania má užívateľ dosah na aktuálne informácie, no zároveň grafické rozhranie a pamäť systému nie je zaťažená zbytočnými dátami. Navyše trieda `AdapterLogger` obsahuje inštanciu triedy `MessageTracker`, ktorá zaznamenáva a informuje užívateľa o počtoch odoslaných a čakajúcich správ. Príklady výpisov detailnej a výkonnostnej simulácie sa nachádzajú v prílohách D a E tohto dokumentu.



Obrázok 4.2 - Graf odozvy servera.

Ďalšou triedou zhromažďujúcou informácie o emulácii je trieda `ResponseTracker`, ktorá spracováva časy odoziev servera na jednotlivé správy. Každá odozva je reprezentovaná triedou `Response` a uchováva časovú pečiatku svojho vytvorenia a samotný čas odozvy v milisekundách.

Tieto odozvy môžu byť ukladané na neskoršie použitie, napríklad vykreslenie grafu (viď Obrázok 4.2), alebo použité pri výpise logovacích správ. Navyše trieda vypočítava maximálny počet správ odoslaných za jednu sekundu a aktuálny počet správ odoslaných za jednu sekundu, ktoré zobrazuje v priradenej oblasti aplikácie (viď Obrázok 3.13).

4.3.2 Komunikácia so serverom

Komunikácia prebieha v dvoch spojeniach (viď Kapitola 2.5.1). Dočasné spojenie poskytuje trieda `ServerController`. Táto trieda uchováva model s informáciami o serveri, na ktorý sa daný adaptér pripája, a poskytuje metódy na vytvorenie TLS certifikátu a odoslanie správy s dátami senzora. Taktiež vďaka triede `ResponseTracker` (viď Kapitola 4.3.1) zaznamenáva čas odozvy servera na jednotlivé správy. Medzi najdôležitejšie metódy triedy `ServerController` patria:

- `createSSLCertificate` - pomocou certifikačnej autority uloženej v zložke aplikácie vytvorí nový certifikát potrebný na bezpečnú komunikáciu so serverom,
- `sendMessage` – metóda vytvorí zabezpečené spojenie, odošle správu s dátami senzoru na server a prijíma odpoveď servera, ktorú zasiela adaptéru na spracovanie. Navyše zaznamenáva čas začiatku a konca komunikácie, aby bolo možné odvodiť čas odozvy servera.

Stále spojenie so serverom poskytuje trieda `ServerReceiver`. Keďže stála sieťová komunikácia je blokujúca operácia, ktorá by zastavila grafické rozhranie aplikácie, trieda `ServerReceiver` vytvára nové vlákno, ktoré riadi. Spojenie je naviazané na činnosť adaptéra. Ak je činnosť povolená, zašle sa registračná správa a prijme sa odpoveď servera. Po kladnej odpovedi sa vlákno prepína do stavu prijímania správ od servera na schránke spojenia vytvoreného registračnou správou. Pri zastavení činnosti adaptéra sa spojenie ukončí. Medzi najdôležitejšie metódy triedy `ServerReceiver` patria:

- `createSSLCertificate` – rovnaká funkcia ako pri triede `ServerController`,
- `run` - metóda rozširujúca rozhranie vlákna `Thread`. Cyklicky prechádza stav spojenia, podľa ktorého vytvára spojenie, počúva na porte alebo uspí toto vlákno,
- `sendMessage` - odoslanie registračnej správy a zaslanie odpovede na adaptér, ktorý ju spracuje,
- `listenForMessages` – prepnutie do stavu prijímania správ od servera,
- `enable` – povolenie na vytvorenie stáleho spojenia, upozorní vlákno, ak je uspaté,
- `disable` – zakázanie spojenia, preruší spojenie a uspí vlákno.

Pri vytváraní a spracovávaní správ komunikácie sa využívajú jednotlivé implementácie rozhrania `Protocol`, vďaka ktorým nie je nutné pri zmene protokolu (viď Kapitola 2.5.3) zasahovať do logiky emulácie, ale stačí vytvoriť novú triedu, ktorá toto rozhranie implementuje.

4.3.3 Emulovanie adaptéra

Prvou a základnou triedou emulovaného adaptéra je trieda `AdapterController` slúžiaca na vykonávanie operácií nad informáciami o adaptéri, ktoré sú uložené v triede `Adapter`. Ďalej slúži na uchovávanie zoznamu inštancií triedy `SensorController`, teda vytvorených senzorov, a ponúka metódy na ukladanie správ od týchto senzorov. Tieto správy, reprezentované triedou `OutMessage`, sú neskôr odosielané na server pomocou rozhrania `Scheduler`. Taktiež uchováva inštancie triedy `ServerController` a `ServerReceiver` obsahujúce prostriedky na pripojenie k danému serveru. Medzi najdôležitejšie metódy triedy `AdapterController` patria:

- `createSensor` - metóda vytvára nový senzor z informácii poskytnutými v parametroch a taktiež v závislosti na parametroch vytvára grafickú reprezentáciu senzora,
- `sendRegisterMessage` - metóda vytvorí registračnú správu (viď Kapitola 2.5.3) a uloží ju do rady správ čakajúcich na odoslanie na prvé miesto,
- `sendMessage` - metóda pridáva novú správu vytvorenú senzorom do rady správ čakajúcich na spracovanie a upozorní užívateľa o vytvorení novej správy pridaním výpisu do sekcie čakajúcich správ logovacej oblasti,
- `messageSuccessfullySent` - metóda odstráni úspešne odoslanú správu z rady čakajúcich správ a upozorní užívateľa pridaním výpisu do logovacej oblasti aplikácie,
- `enable` - metóda upozorní adaptér, že sa môže pripojiť na server a začať spracovávať čakajúce správy, ak existujú,
- `disable` - metóda upozorní adaptér, aby sa odpojil od servera a pozastavil spracovávanie čakajúcich správ,
- `changeValueOnSensor` - táto metóda sa využíva po prijatí správy o zmene stavu aktuátora od servera, aby upozornila senzor, ktorý túto hodnotu obsahuje,
- `saveToXml` - metóda vytvorí XML súbor s informáciami o adaptéri, jeho vytvorených senzoroch a emulovaných hodnotách.

Ďalej nasleduje rozhranie plánovača `Scheduler`, ktorého implementácie `DetailedScheduler` a `PerformanceScheduler` zodpovedajú za cyklické spracovávanie a odosielanie čakajúcich správ uložených na adaptéri. Tieto triedy vytvárajú nové vlákno, aby nezaťažovali vlákno grafického rozhrania a zároveň mohli spracovávať správy v čo najkratšom čase. Trieda `DetailedScheduler`, používaná pri detailnej simulácii (viď Kapitola 3.3), využíva oba druhy spojení so serverom, pričom výber spojenia závisí na druhu správy (viď Kapitola 4.3.2). Na druhú stranu vo výkonnostnej simulácii (viď Kapitola 3.4), kde sa naraz zasielajú správy stoviek až tisícov adaptérov, je využívané iba dočasné spojenie. Stále spojenie, vytvárajúce druhé vlákno pre každý adaptér, by nežiaduco zaťažovalo zariadenie, na ktorom aplikácia beží. Navyše pri testovaní serverovej časti architektúry, na ktoré je výkonnostná simulácia vytvorená, nie je potrebné zasielať adaptérom správy, teda stále spojenie nemusí byť použité. Obe implementované triedy sa navyše starajú o výpisy časov odoziev servera na jednotlivé požiadavky do logovacej sekcie adaptéra a oznamovanie o nesprávnych, či porušených správach od servera. Ak plánovač nepreberie od adaptéra žiadnu novú správu, uspí vlákno, čím znižuje zaťaženie systému. Po pridaní novej správy adaptér obnoví činnosť plánovača.

4.3.4 Emulovanie senzorov

Trieda `SensorController` disponuje metódami potrebnými na emulovanie senzorov a uchováva informácie o danom senzore v triede `Server`. Pre napodobenie uspania senzoru je každej inštancii triedy `SensorController` vytvorený vlastný časovač nastavený na interval uspania daný užívateľom, ktorým je senzor riadený. V čase vypršania časovača senzor vytvorí správu pomocou zadanej implementácie rozhrania `Protocol`, ktorú uloží na adaptér. Interval uspania môže byť počas behu aplikácie získaný a aktualizovaný z odpovede servera.

Jednou z informácií o senzore je zoznam emulovaných veličín. Každá veličina z tabuliek typov (viď Tabuľka 1 a Tabuľka 2) je reprezentovaná vlastnou implementáciou rozhrania `Value` a rozširuje triedu `AbstractValue<T>`, ktorá poskytuje implementáciu metód používaných všetkými typmi veličín. Konštrukcia triedy `Class<T>` znamená, že trieda využíva generické typy, kde `T` je dátový typ definovaný rozširujúcou triedou. [11] Vďaka tomuto dokážu jednotlivé rozšírenia triedy

`AbstractValue` používať rôzne dátové typy pre emulované hodnoty veličín, pričom definícia metódy zostáva len jedna. Napríklad pri metóde `public T getValue()` definovanej v triede `AbstractValue` a využívanej všetkými implementáciami emulovaných veličín, sa dosadzuje dátový typ hodnoty `T` z rozširujúcej triedy pri kompilácií. Touto konštrukciou som docielil možnosť využívania metód jedného rozhrania pre rôzne dátové typy, a taktiež jednoduchšie pridávanie nových typov veličín v budúcnosti.

Každá emulovaná veličina navyše určuje typy generátorov, ktoré môžu byť použité, implementovaním rozhraní `HasNormalDistribution`, `HasLinearDistribution`, `HasBooleanRandom`. Samotné generovanie nastáva v polovici intervalu uspania, čím sa znižuje záťaž systému a zároveň je možnosť vidieť novú hodnotu pred odoslaním na server.

Rozhranie `HasNormalDistribution` obsahuje metódy pre generovanie nových hodnôt pomocou normálneho (Gaussovho) rozloženia (viď Kapitola 3.5), pričom užívateľ pri vytváraní senzora poskytne maximálnu a minimálnu možnú hodnotu, strednú hodnotu a smerodajnú odchýlku. Na implementáciu tohto rozloženia som použil metódu `nextGaussian` triedy `Random`, základného balíčka `java.util` jazyku Java. Táto funkcia vracia pseudonáhodné číslo so strednou hodnotou 0.0 a smerodajnou odchýlkou 1.0. Toto číslo ďalej upravím nasledujúcim algoritmom, aby vyhovoval požadovanému rozsahu.

Algoritmus výpočtu generovanej hodnoty podľa normálneho rozloženia:

1. výsledná hodnota sa rovná návratová hodnota funkcie `nextGaussian` vynásobená zadanou smerodajnou odchýlkou a pripočítanou zadanou strednou hodnotou
2. ak je výsledná hodnota väčšia ako zadané maximum alebo menšia ako zadané minimum, vráť sa na krok 1

Rozhranie `HasLinearDistribution` obsahuje metódy pre generovanie nových hodnôt pomocou lineárnej funkcie (viď Kapitola 3.5). Užívateľ poskytuje maximálnu, minimálnu možnú hodnotu a konštantu kroku algoritmu.

Algoritmus výpočtu generovanej hodnoty pomocou lineárnej funkcie:

1. výsledná hodnota sa rovná aktuálna hodnota meranej veličiny, ku ktorej je pripočítaná konštanta zadaná užívateľom
2. ak je výsledná hodnota väčšia ako maximum, výslednou hodnotou je maximum
3. ak je výsledná hodnota menšia ako minimum, výslednou hodnotou je minimum

Rozhranie `HasBooleanRandom` ponúka metódy, vďaka ktorým sa pseudonáhodne mení hodnota veličín určujúcich pravdivosť[2]. Pri vytváraní senzoru užívateľ zadáva pravdepodobnosť zmeny hodnoty. Na implementáciu tejto funkcie som zvolil metódu `nextDouble` triedy `Random`, základného balíčka `java.util` jazyku Java. Táto funkcia generuje pseudonáhodné číslo z rozsahu 0.0 až 1.0, ktoré porovnávam so zadanou pravdepodobnosťou. Ak je pravdepodobnosť väčšia alebo rovná vygenerovanému číslu, hodnota sa zmení, inak zostáva rovnaká.

4.4 Úloha výkonnostnej simulácie

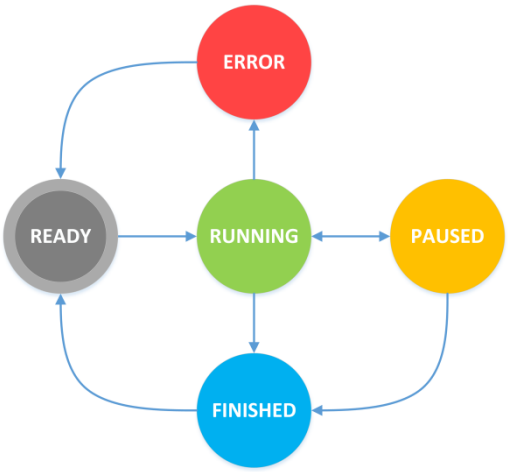
Úloha výkonnostnej simulácie (viď Kapitola 3.4) je reprezentovaná triedou `SimulationTask`. Základné nastavenia úlohy zadané pri jej vytváraní sa ukladajú do tried `TaskParameters`

[2] Hodnoty dátového typu boolean zahrnuté v komunikačnom protokole (viď Kapitola 2.5.3)

a `ValueParameters` a slúžia ako zdroj informácií pre generovanie adaptérov a ich senzorov po spustení danej úlohy. Medzi ďalšie nastavenia patrí určenie ukončovacích podmienok úlohy pomocou triedy `StopCondition`, ktorá o naplnení jednotlivých podmienkach rozhoduje na základe informácií uložených v triedach `AdapterLogger` a `StopWatch`.

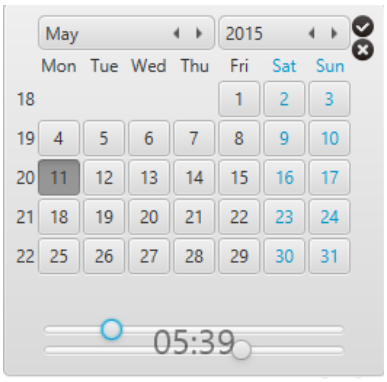
Riadenie úlohy prebieha v závislosti na jej aktuálnom stave, pričom úloha môže nadobúdať jeden z piatich stavov (viď Obrázok 4.3):

- `READY` - vytvorená úloha je pripravená na spustenie,
- `RUNNING` - úloha vytvorila adaptéry s príslušnými senzormi a jednotlivé adaptéry začínajú zasielať dáta na server,
- `PAUSED` - vytvorené adaptéry sú odpojené od servera a nezasielajú žiadne dáta,
- `FINISHED` - úloha bola manuálne ukončená alebo bola splnená jedna z ukončovacích podmienok, objekty adaptérov sú odstránené z emulátora, a taktiež z databáze,
- `ERROR` – za behu úlohy sa vyskytla závažná chyba, objekty adaptérov sú odstránené z emulátora, a taktiež z databáze.



Obrázok 4.3 - Prechody stavov úlohy.

Trieda `QueueWatcher` pracuje nad zoznamom vytvorených úloh a zabezpečuje automatické spracovávanie týchto úloh. V závislosti na nastaveniach užívateľa môže spustiť spracovávanie okamžite alebo v prednastavený čas (viď Obrázok 4.4), ktorý je kontrolovaný samostatným časovačom. Aby mohla byť úloha zo zoznamu spracovaná automaticky, musí mať aspoň jednu ukončovaciu podmienku a zároveň musí byť v stave `READY`. Úlohy, ktoré sa nachádzajú v inom stave sú ignorované.



Obrázok 4.4 - Výber času automatického spracovania úloh.

4.5 Test maximálneho počtu vlákien

Operačné systémy štandardne obmedzujú počet vlákien, ktoré môže proces vytvoriť. Táto skutočnosť môže obmedziť počet vytváraných adaptérov a senzorov pri výkonnostnej simulácii, kde každý adaptér vytvára jedno vlákno pre svoj plánovač (viď Kapitola 4.3.3) a zároveň každý senzor vytvára vlákno v implementácii svojho časovača. Preto som vytvoril jednoduchú pomôcku, spúšťanú z výberového okna aplikácie (viď Obrázok 3.3), ktorá umožňuje otestovať koľko vlákien dokáže daný systém vytvoriť. Keďže sa testujú hraničné možnosti systému, test je náročný na zdroje a môže dočasne obmedziť plynulý chod systému. Testovaním operačných systémov použitých na vývoj emulátora (viď Kapitola 4), ktorým neboli zmenené štandardné nastavenia, boli namerané hodnoty:

- Windows 8 64bit – 104000 vlákien na jeden proces. Nastavenia možné upraviť pomocou parametrov JVM (Java Virtual Machine) zadaných pri spúšťaní aplikácie,
- Fedora 20 „HeisenBug“ 64bit – 1024 vlákien na jeden proces. Nastavenia možné upraviť pomocou systémovej funkcie `ulimit`[3].

4.6 Testovanie

Vývoj emulátora bol úzko spätý s vývojom serverovej aplikácie, a preto prebiehalo testovanie od skorých fáz implementácie. Postupom času sa vďaka mojej aplikácii podarilo lokalizovať chyby servera, ako napríklad chyby v komunikačnom protokole, chyba v nastaveniach tabuliek databáze, chyba pri prijímaní dát z veľkého množstva adaptérov a ďalšie.

Pri testovaní sa emulátor pripájal na aktuálnu verziu serverovej aplikácie dostupnej na dedikovaných serveroch `ant-1.fit.vutbr.cz`[4] a `ant-2.fit.vutbr.cz`[5] výskumnej skupiny a bol spúšťaný na operačných systémoch *Windows 8 64-bit* a linuxovej distribúcii *Fedora 20 „HeisenBug“ 64-bit*.

4.6.1 Testovanie detailnej simulácie

Pri testovaní detailnej simulácie (viď Kapitola 3.3) bola výsledná aplikácia spolu s návodom na použitie poskytnutá členovi výskumnej skupiny, ktorý je súčasťou vývoja koncovej ovládacej stanice, konkrétne aplikácie pre platformu *Android*. Užívateľ mal za úlohu vytvoriť niekoľko adaptérov a senzorov, pričom požiadavky boli:

- Pri vytváraní adaptérov kontrolovať duplicitu identifikačného čísla v databáze pomocou funkcií emulátora,
- Vo vytvorených senzoroch použiť všetky implementované emulované typy veličín a typy generátorov nových hodnôt,
- Uložiť a neskôr načítať vytvorené adaptéry,
- Pridať vytvorené adaptéry do prostredia koncovej ovládacej stanice,
- Pomocou koncovej ovládacej stanice zmeniť intervaly uspávania na vytvorených senzoroch,
- Pomocou koncovej ovládacej stanice zmeniť stav hodnoty aktuátora,

[3] Definícia funkcie dostupná z manuálových stránok distribúcie Fedora 20
http://linuxmanpages.net/manpages/fedora20/man1/bash.1.html
[4] Konfigurácia stroja: Intel(R) Xeon(R) CPU E5410 @ 2.33GHz s operačnou pamäťou veľkosti 12GB
[5] Konfigurácia stroja: Intel(R) Xeon(R) CPU E5410 @ 2.33GHz s operačnou pamäťou veľkosti 10GB

- Zmazať niekoľko vybraných senzorov,
- Zmazať vybraný adaptér.

Užívateľ bol s užívateľským rozhraním veľmi spokojný. Návrh aplikácie bol dobrý a užívateľ si rýchlo zvykol na prácu s aplikáciou. Po prečítaní návodu dokázal splniť všetky zadané požiadavky.

Behom pridávania emulovaných typov veličín sa však zistilo, že aplikácia pri dvoch z nich nežiaduco zobrazuje hodnoty v užívateľskom rozhraní. Ďalšia zistená chyba bola pri odstraňovaní senzorov, kedy sa dané senzory nesprávne odstránili z databáze. Tieto chyby boli zaznamenané a následne opravené.

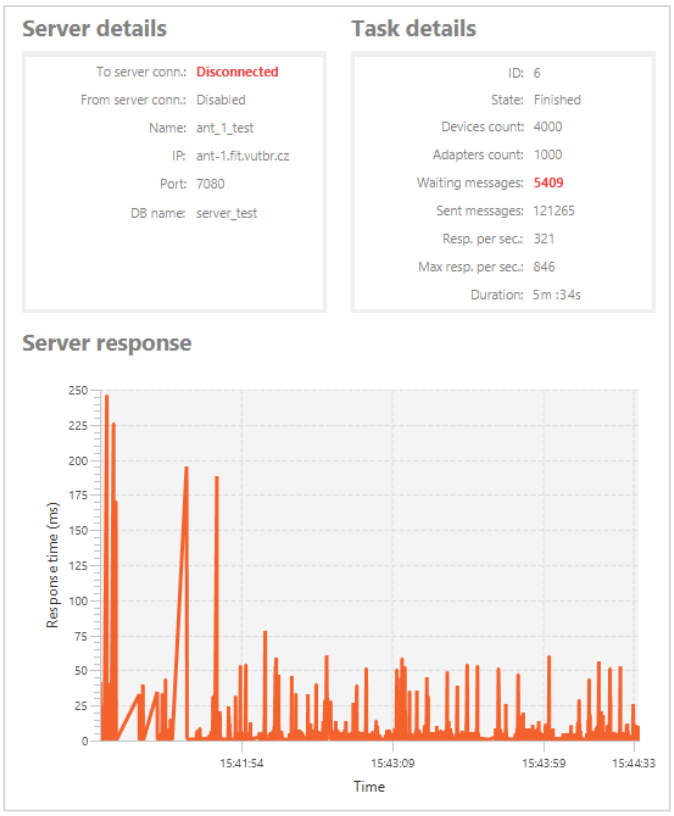
Užívateľ nahlásil iba jednu ďalšiu výhradu, a to zahrnutie vysvetliviek jednotlivých logovacích výpisov, hlavne chybových, do užívateľskej príručky.

4.6.2 Testovanie výkonnostnej simulácie

Pri testovaní výkonnostnej simulácie som sa zameriaval na zistenie hraničných možností emulátora a zároveň serverovej aplikácie. Všetky testy prebiehali na operačnom systéme *Windows 8 64bit* a pozostávali z postupného pridávania úloh so zvyšujúcim sa počtom adaptérov a senzorov s jednou meranou veličinou. Pri všetkých úlohách som zvolil rozsah intervalu uspania senzorov v rozmedzí 5 až 10 sekúnd, vďaka čomu som chcel dosiahnuť skoro stálu nutnosť komunikácie so serverom, a tým pádom vytvoriť záťaž na serverovú aplikáciu. Počty emulovaných adaptérov a senzorov v jednotlivých úlohách boli nasledovné:

1. Úloha – 100 adaptérov, každý adaptér 1 senzor
2. Úloha – 300 adaptérov, každý adaptér 1 senzor
3. Úloha – 500 adaptérov, každý adaptér 1 senzor
4. Úloha – 500 adaptérov, každý adaptér 2 senzory
5. Úloha – 1000 adaptérov, každý adaptér 2 senzory
6. Úloha – 1000 adaptérov, každý adaptér 4 senzory

Vykonávanie úloh 1 až 4 prebiehalo bez známky chýb na strane emulátora i servera a užívateľské prostredie bolo plne reagujúce na manuálne ovládanie úloh. Pri 5. úlohe som začal zaznamenávať chyby na strane servera, ktoré boli spôsobené odmietnutím niekoľkých spojení pri väčšom zhluku odosielaných správ, no úloha pokračovala bez ďalších závažných chýb. Pri manuálnom ovládaní úlohy som zaznamenal mierne oneskorenie v reakcii aplikácie na stlačenie tlačidla. Po niekoľkých minútach behu 6. úlohy sa počet odmietnutých spojení serverom začal značne zvyšovať, čo neskôr vyústilo až k pádu serverovej aplikácie. Napriek tomu dokázal emulátor vytvoriť a odoslať viac ako 120,000 správ, pričom maximálny počet odoslaných správ za 1 sekundu bol 846 (viď Obrázok 4.5). Taktiež je na grafe vidieť výchylky v odozvách spojení spôsobené odmietnutými spojeniami. V priebehu úlohy dokázala aplikácia vykresľovať graf a zároveň štatistické údaje plynule, avšak manuálne ovládanie vykazovalo dlhú reakciu na stlačenie tlačidiel, pretože väčšina systémových zdrojov bola zaneprázdnená ošetrovaním chybných spojení. Tento nedostatok je možné čiastočne obmedziť optimalizáciou operácií vykonávaných grafickým vláknom aplikácie. Výkonnostná simulácia splnila požiadavky na zistenie hraničných možností serverovej aplikácie, pričom možnosti emulátora neboli naplno využité. Ďalšie testovanie so zvyšujúcim sa počtom emulovaných adaptérov môže pokračovať až po optimalizácii serverovej aplikácie.



Obrázok 4.5 - Výsledky 6. testovanej úlohy.

5 Záver

Cieľom tejto bakalárskej práce bolo vytvoriť aplikáciu emulujúcu koncové prvky inteligentnej domácnosti, ktorá by dokázala dočasne nahradiť nutnosť použitia reálnych prvkov pri vývoji a testovaní zvyšných častí systému. Pre splnenie tejto úlohy som musel podrobne naštudovať princípy fungovania architektúry systému, existujúcich prvkov a vnútro systémovej komunikácie. Ďalej som sa riadil zásadami a postupmi pre vytvorenie užívateľsky prívetivého rozhrania, ktoré spríjemňuje prácu s aplikáciou a umožňuje efektívne vykonávanie častých operácií. Na základe týchto poznatkov som vytvoril návrh a implementoval aplikáciu, ktorej časti sa zameriavajú na rôzne vrstvy architektúry systému inteligentnej domácnosti, a tým poskytuje širšie spektrum možnej využiteľnosti. Prvá časť, zameraná na testovanie koncových ovládacích staníc, dokáže podrobne zobraziť emulované prvky, ich merané veličiny a taktiež poskytuje možnosť ich samostatného ovládania. Druhá časť aplikácie, zameraná na serverovú vrstvu architektúry, dokáže zabezpečiť rýchle vytvorenie a automatické ovládanie veľkého počtu emulovaných prvkov (v jednotkách stoviek až tisícov), vďaka čomu umožňuje simuláciu záťaže na server. Vďaka emulátoru sa dokázalo lokalizovať mnohé chyby serverovej aplikácie už počas jeho implementácie, vďaka čomu preukázal svoju užitočnosť. V neposlednom rade na základe pozitívnych názorov užívateľov môžem konštatovať, že aplikácia splnila všetky predsavzaté požiadavky.

V rámci tejto práce bola vyvinutá aj jednoduchá samostatná konzolová aplikácia, ktorá slúži ako server pre emulátor a spúšťa sa na serveri inteligentnej domácnosti. Keďže prístup k databáze je z bezpečnostných dôvodov obmedzený na lokálne adresy, táto aplikácia poskytuje emulátoru prístup k požadovaným dátam. Komunikácia prebieha na základe jednoduchého *XML* protokolu a sú poskytnuté funkcie ako kontrola duplicity identifikátorov, zaslanie zoznamu všetkých existujúcich adaptérov a zmazanie daného adaptéra, poprípade senzora.

Ďalším pokračovaním vývoja emulátora by mohla byť synchronizácia úloh výkonnostnej simulácie viacerých emulátorov. V prípade, že by nestačila záťaž na serverovú aplikáciu generovaná na jednom zariadení, užívateľ by spustil emulátor na viacerých zariadeniach a synchronizoval by ich úlohy pomocou servera emulátora. Tieto úlohy by boli ovládané z jednej (master) inštancie emulátora. Týmto spôsobom by bolo možné vytvoriť niekoľkonásobnú záťaž na server, pričom samotná záťaž na systémové zdroje zariadení so spustenými emulátormi by sa znížila. Ako rozšírenie detailnej simulácie (viď Kapitola 3.3) by sa mohla implementovať logika prepojenia jednotlivých emulovaných veličín na rovnakom senzore. Táto logika by zahrňovala možné stavy emulovaných veličín a akcie spustené pri zmene tohto stavu. Napríklad pri zmene stavu aktuátora užívateľom by začala rásť hodnota teploty, vďaka čomu by bolo možné simulovať zopnutie vyhrievania domácnosti.

Literatúra

[1] GLOBAL STRATEGY AND BUSINESS DEVELOPMENT, FREESCALE AND EMERGING TECHNOLOGIES, ARM. 2014. *What the Internet of Things (IoT) Needs to Become a Reality* [online]. Rev 2. [cit. 2015-02-03]. Dostupné z: http://cache.freescale.com/files/32bit/doc/white_paper/INTOTHNGSWP.pdf

[2] *Stránky skupiny "Internet of Things"* [online]. 2014. [cit. 2015-02-17]. Dostupné z: https://ant-2.fit.vutbr.cz/projects/iot/wiki

[3] Transport Layer Security. 2015. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, last modified 04:52, 07.02.2015 [cit. 2015-03-03]. Dostupné z: http://en.wikipedia.org/wiki/Transport_Layer_Security

[4] XML TECHNOLOGY. 2015. *World Wide Web Consortium (W3C)* [online]. [cit. 2015-03-03]. Dostupné z: http://www.w3.org/standards/xml/

[5] PERINGER, Petr. 2006. *Modelování a simulace: Studiní opora* [online]. FIT VUT v Brně [cit. 2015-04-14]. Dostupné z: https://wis.fit.vutbr.cz/FIT/db/dir.php/course/IMS-IT/texts/.cs

[6] LIPTÁK, Juraj. *Nástroj pro testování souborů náhodných čísel*. Brno, 2011. Bakalářská práce. FIT VUT v Brňe. Vedoucí práce Ing. Martin Hrubý, Ph.D. Dostupné z: http://www.fit.vutbr.cz/study/DP/BP.php.cs?id=11737&file=t

[7] What's New in JDK 8. *Oracle* [online]. [cit. 2015-03-14]. Dostupné z: http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html

[8] Using JavaFX Properties and Binding. 2013. *Client Technologies: Java Platform, Standard Edition (Java SE) 8 Release 8* [online]. [cit. 2015-03-14]. Dostupné z: http://docs.oracle.com/javafx/2/binding/jfxpub-binding.htm

[9] .properties. 2015. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, last modified 18:59, 17.04.2015 [cit. 2015-04-18]. Dostupné z: http://en.wikipedia.org/wiki/.properties

[10] FUSEK, Zdeněk. *Grafické uživatelské prostředí v JavaFX*. Brno, 2013. Bakalářská práce. FEKT VUT v Brňe. Vedoucí práve Ing. Jan Karásek. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=69191

[11] Generic Types. 2015. *The Java™ Tutorials* [online]. [cit. 2015-04-07]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/generics/types.html

Zoznam príloh

Príloha A - Šablóna súboru s nastaveniami aplikácie

Ak pri spustení aplikácie tento súbor neexistuje, je automaticky vytvorený nový so štandardnými nastaveniami.

```
#emulator version
version=2.0.6
#predefined servers
#separated by "," -> after adding new sensor ( for example: tmp), please add new properties:
#   tmp_server_ip=1.1.1.1
#   tmp_server_port=1111
#   tmp_server_db=dbName
servers=ant_devel,ant_production
#devel_new
ant_devel_server_ip=ant-2.fit.vutbr.cz
ant_devel_server_port=7080
ant_devel_server_db=home5
#ant_production
ant_production_server_ip=ant-2.fit.vutbr.cz
ant_production_server_port=9092
ant_production_server_db=home4prod
#Adapter default firmware version
defaultFirmware=0
#Adapter logger
#ALL,TRACE,DEBUG,INFO,WARN,ERROR,FATAL,OFF
logLevel=INFO
```

Príloha, Obrázok 1 – Súbor s nastaveniami aplikácie.

Príloha B - Záznam súborov vytváraných aplikáciou

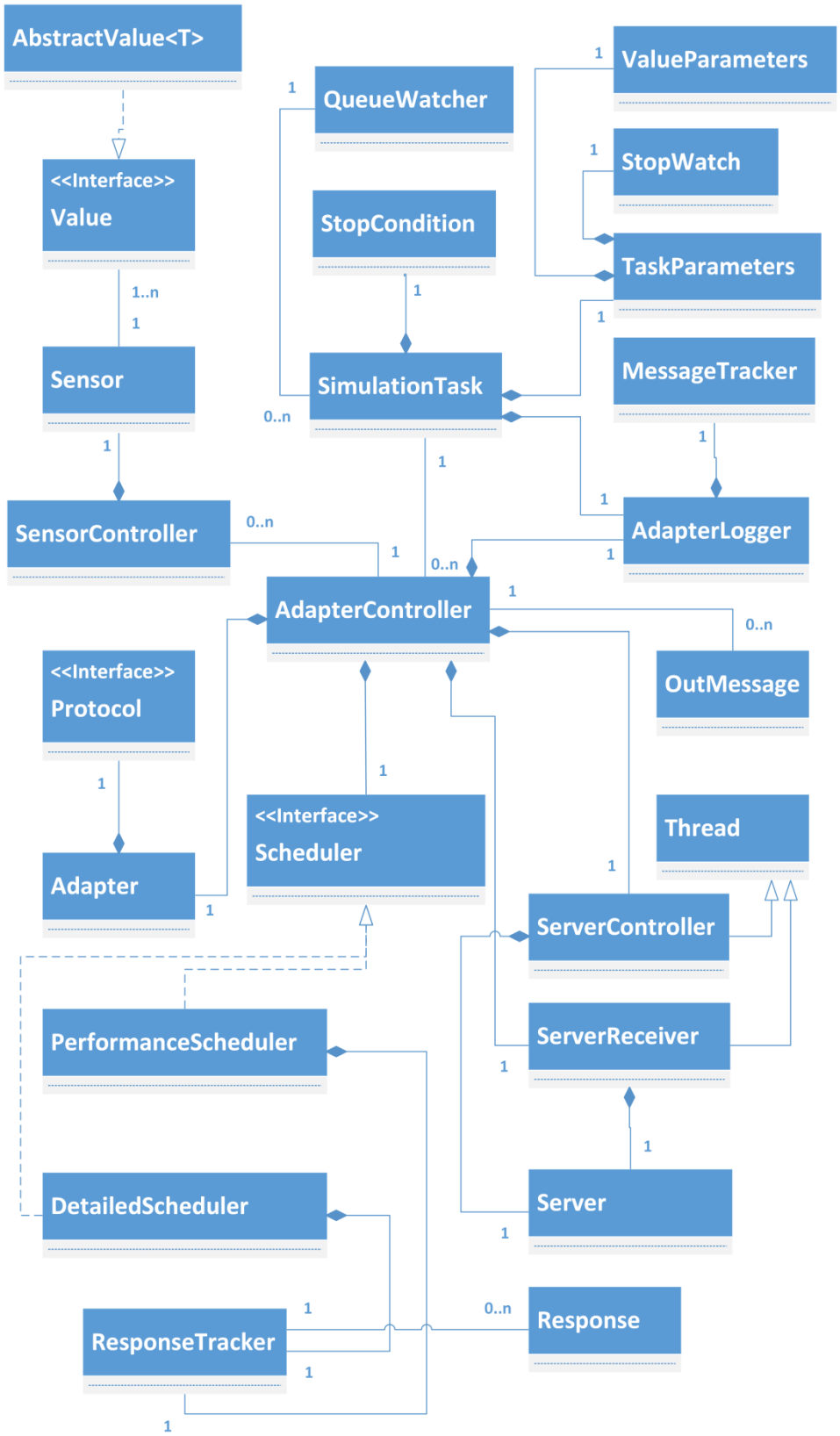
```xml
<?xml version="1.0" encoding="UTF-8"?>
<adapters>
    <adapter firmware="0.0" id="99999" name="Test adapter #1"
        protocol="0.1" registered="false">
        <server db="home5" ip="ant-2.fit.vutbr.cz" name="ant_devel" port="7080"/>
        <sensors>
            <sensor id="99991">
                <name>Multisensor #1</name>
                <refresh>10</refresh>
                <signal>81</signal>
                <battery>90</battery>
                <icon>Multisensor</icon>
                <header_color>#FFE037</header_color>
                <values>
                    <value generate_value="true" name="Temperature" store_history="false"
                     type="Temperature">
                        <initial_value>25.5</initial_value>
                        <generator seed="1431116709174" type="ND">
                            <params avg="25.5" dev="0.5" max="50.0" min="-50.0"/>
                        </generator>
                    </value>
                    <value generate_value="false" name="Humidity" store_history="false"
                     type="Humidity">
                        <initial_value>55</initial_value>
                    </value>
                    <value generate_value="true" name="On/Off Actuator" store_history="false"
                     type="On/Off Actuator">
                        <initial_value>false</initial_value>
                        <generator seed="1431116722197" type="BR">
                            <params probability="0.42"/>
                        </generator>
                    </value>
                </values>
            </sensor>
            <sensor id="99992">
                <name>Multisensor 1</name>
                <refresh>5</refresh>
                <signal>90</signal>
                <battery>95</battery>
                <icon>Pressure</icon>
                <header_color>#0CDF56</header_color>
                <values>
                    <value generate_value="false" name="On/Off Sensor" store_history="false"
                     type="On/Off Sensor">
                        <initial_value>false</initial_value>
                    </value>
                    <value generate_value="false" name="Light" store_history="false" type="Light">
                        <initial_value>400.0</initial_value>
                    </value>
                    <value generate_value="false" name="B.O. temperature" store_history="false"
                        type="B.O. temperature">
                        <initial_value>25.5</initial_value>
                    </value>
                    <value generate_value="false" name="Boiler status" store_history="false"
                        type="Boiler status">
                        <initial_value>1</initial_value>
                    </value>
                </values>
            </sensor>
        </sensors>
    </adapter>
</adapters>
```

Príloha, Obrázok 2 – Adaptér uložený detailnou simuláciou.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tasks>
    <task>
        <server db="home5" ip="ant-2.fit.vutbr.cz" name="ant_devel" port="7080"/>
        <parameters ada_count="500" protocol="0.1" start_id="1000">
            <sensors_count max="20" min="1" seed="1431117686219"/>
            <refresh_time max="10" min="5" seed="1431117686219"/>
            <save_dir>logs/performance</save_dir>
            <enabled_values seed="1431117686219">Humidity,Pressure,
                Light,Noise,Emissions,Temperature</enabled_values>
        </parameters>
        <stop_conditions>
            <time_duration enabled="true" sec="3722"/>
            <sent_messages count="100000" enabled="true"/>
            <waiting_messages count="5000" enabled="true"/>
        </stop_conditions>
    </task>
    <task>
        <server db="home4prod" ip="ant-2.fit.vutbr.cz" name="ant_production" port="9092"/>
        <parameters ada_count="200" protocol="0.1" start_id="1000">
            <sensors_count max="10" min="10" seed="1431117774967"/>
            <refresh_time max="5" min="5" seed="1431117774967"/>
            <save_dir>logs/performance</save_dir>
            <enabled_values seed="1431117774967">Humidity,Light,
                Emissions,Temperature</enabled_values>
        </parameters>
        <stop_conditions>
            <time_duration enabled="false" sec="0"/>
            <sent_messages count="0" enabled="false"/>
            <waiting_messages count="5000" enabled="true"/>
        </stop_conditions>
    </task>
</tasks>
```

Príloha, Obrázok 3 – Úloha uložená výkonnostnou simuláciou.

Príloha C – Objektový návrh jadra aplikácie



Príloha, Obrázok 4 – Diagram tried objektového návrhu.

Príloha D - Výpisy detailnej simulácie

Príklad logovacích výpisov jedného adaptéra v detailnej simulácii. Na začiatku dokumentu vidieť proces vytvárania stáleho spojenia zaslaním registračnej správy a nasledovného počúvania na správy od servera. Taktiež sú v dokumente výrazne odlíšené logovacie sekcie.

```
== ADAPTER LOG ===============================================
23:04:38(932) - Server receiver paused
23:04:44(256) - Test adapter #1/99999 -> Registering adapter -> 0ms
23:04:44(256) - Server receiver listening
23:04:54(509) - Sensor Multisensor #2/99992 trying to send message.
23:04:54(613) - Sensor Multisensor #2/99992 --> data sent -> 13ms
23:04:59(510) - Sensor Multisensor #2/99992 trying to send message.
23:04:59(618) - Sensor Multisensor #2/99992 --> data sent -> 12ms
23:05:03(026) - Sensor Multisensor #1/99991 trying to send message.
23:05:03(137) - Sensor Multisensor #1/99991 --> data sent -> 12ms
23:05:03(138) - Sensor Multisensor #1/99991 new Refresh time --> 5
23:05:04(511) - Sensor Multisensor #2/99992 trying to send message.
23:05:04(612) - Sensor Multisensor #2/99992 --> data sent -> 12ms
23:05:09(511) - Sensor Multisensor #2/99992 trying to send message.
23:05:09(612) - Sensor Multisensor #2/99992 --> data sent -> 12ms
23:05:10(637) - Sensor Multisensor #1/99991 trying to send message.
23:05:10(740) - Sensor Multisensor #1/99991 --> data sent -> 12ms
23:05:14(512) - Sensor Multisensor #2/99992 trying to send message.
23:05:14(613) - Sensor Multisensor #2/99992 --> data sent -> 12ms
23:05:20(639) - Sensor Multisensor #1/99991 trying to send message.
23:05:20(740) - Sensor Multisensor #1/99991 --> data sent -> 12ms
23:05:24(513) - Sensor Multisensor #2/99992 trying to send message.
23:05:24(613) - Sensor Multisensor #2/99992 --> data sent -> 13ms
23:05:25(640) - Sensor Multisensor #1/99991 trying to send message.
23:05:25(739) - Sensor Multisensor #1/99991 --> data sent -> 12ms
23:05:29(515) - Sensor Multisensor #2/99992 trying to send message.
23:05:29(965) - Sensor Multisensor #2/99992 --> data sent -> 13ms
23:05:30(641) - Sensor Multisensor #1/99991 trying to send message.
23:05:30(746) - Sensor Multisensor #1/99991 --> data sent -> 12ms
23:05:34(515) - Sensor Multisensor #2/99992 trying to send message.
23:05:34(619) - Sensor Multisensor #2/99992 --> data sent -> 12ms
23:05:35(642) - Sensor Multisensor #1/99991 trying to send message.
23:05:35(744) - Sensor Multisensor #1/99991 --> data sent -> 12ms
23:05:39(516) - Sensor Multisensor #2/99992 trying to send message.
23:05:39(617) - Sensor Multisensor #2/99992 --> data sent -> 13ms
23:05:40(642) - Sensor Multisensor #1/99991 trying to send message.
23:05:40(743) - Sensor Multisensor #1/99991 --> data sent -> 12ms
23:05:42(783) - Server receiver paused

== TO BE SENT LOG ============================================

== ERROR LOG =================================================
```

Príloha E - Výpisy výkonnostnej simulácie

Príklad logovacích výpisov jednej úlohy výkonnostnej simulácie. Ako je na príklade vidieť, na začiatku súboru je zhrnutie nastavení úlohy nasledované výpismi o vytváraní adaptérov. Taktiež sa tu nachádzajú výrazne označené stavy úlohy a samotné výpisy zasielania dát. Po skončení vykonávania úlohy sú zapísané štatistické údaje, počet správ, ktoré pred ukončením úlohy čakali na odoslanie a chybové hlásenia. V danom prípade emulátor vytvára 20 adaptérov, z ktorých každý má 20 senzorov a interval uspania 10 sekúnd. Výpisy boli pre potrebu tohto dokumentu skrátené.

```
== TASK LOG ================================================
Task parameters:
      Server:
        Name: ant_devel
        IP: ant-2.fit.vutbr.cz
        Port: 7080
        Database: home5
      Adapters:
        Adapters count: 20
        Protocol: 0.1
        Start ID: 1000
        Sensors count per adapter: 20 -> 20
        Refresh time range: 10 -> 10

16:24:46(831) - Estimated task duration: 1m : 0s

16:24:46(832) - Estimated task's sent message count: 10000

16:24:46(832) - Estimated task's waiting messages count: 100000

16:25:02(464) - Creating Adapter/1000 with 20 sensors
16:25:02(473) - Creating Adapter/1001 with 20 sensors
16:25:02(477) - Creating Adapter/1002 with 20 sensors
...
16:25:02(643) - Creating Adapter/1018 with 20 sensors
16:25:02(649) - Creating Adapter/1019 with 20 sensors
16:25:02(657) - Created 260 sensors.
16:25:02(725) - Task RUNNING
16:25:03(084) - Adapter/1005 -> Registering adapter -> 26ms
16:25:03(084) - Adapter/1004 -> Registering adapter -> 28ms
16:25:03(084) - Adapter/1010 -> Registering adapter -> 29ms
...
16:25:03(297) - Adapter/1009 -> Registering adapter -> 22ms
16:25:03(297) - Adapter/1014 -> Registering adapter -> 22ms
16:25:03(403) - Adapter/1012 -> Registering adapter -> 22ms
16:25:17(796) - Adapter/1003 -> Sensor/10031006 --> data sent -> 4ms
16:25:17(806) - Adapter/1000 -> Sensor/10001016 --> data sent -> 1ms
16:25:17(806) - Adapter/1006 -> Sensor/10061011 --> data sent -> 4ms
16:25:17(812) - Adapter/1005 -> Sensor/10051019 --> data sent -> 11ms
16:25:17(812) - Adapter/1010 -> Sensor/10101015 --> data sent -> 11ms
...
```

```
16:25:42(754) - Adapter/1013 -> Sensor/10131031 --> data sent -> 195ms
16:25:42(754) - Adapter/1014 -> Sensor/10141025 --> data sent -> 195ms
16:25:42(768) - Adapter/1016 -> Sensor/10161035 --> data sent -> 12ms
16:25:42(769) - Adapter/1017 -> Sensor/10171026 --> data sent -> 16ms
16:25:42(769) - Adapter/1019 -> Sensor/10191037 --> data sent -> 19ms
16:25:42(783) - Task PAUSED
17:53:15(564) - Task FINISHED
17:53:15(564) - Task duration: 39 s
17:53:15(564) - Max responses per second: 46
17:53:15(564) - Average sent messages per second: 19
Number of sent messages: 773

== TO BE SENT LOG ==============================================
Number of waiting messages: 447

== ERROR LOG ===================================================
16:25:03(091) - Adapter/1018 -> Socket is not valid. Reinitializing
certificate
```